VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Roman Surový

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SIMULACE PRŮMYSLOVÝCH AUTOMATIZAČNÍCH PROTOKOLŮ
INDUSTRIAL AUTOMATION PROTOCOLS SIMULATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Roman Surový**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Blažek**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Roman Surový ***ID:*** 164946
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Simulace průmyslových automatizačních protokolů

POKYNY PRO VYPRACOVÁNÍ:

Práce je zaměřena na realizování pracoviště, které bude simulovat síťovou infrastrukturu průmyslových automatizačních protokolů. Student v rámci práce provede analýzu průmyslových protokolů (doporučeny – ModBus, Profibus/Profinet, EtherCAT, CANopen). Dále bude zprovozněna síťová infrastruktura s PLC a vybranými protokoly. Výstupem práce bude pracoviště, které bude schopno simulovat reálnou komunikaci alespoň dvou vybraných protokolů mezi dvěma a více zařízeními. Dílčí částí práce bude simulace alespoň tří typů zařízeních vyskytujících se v průmyslovém prostředí.

DOPORUČENÁ LITERATURA:

[1] A. Poschraann and P. Neumann, "Architecture and model of Profinet IO," 2004 IEEE Africon. 7th Africon Conference in Africa (IEEE Cat. No.04CH37590), Gaborone, 2004, pp. 1213-1218 Vol.2. doi: 10.1109/AFRICON.2004.1406884

[2] M. Farsi and K. Ratcliff, "An introduction to CANopen and CANopen communication issues," IEE Colloquium on CANopen Implementation (Digest No. 1997/384), London, UK, 1997, pp. 2/1-2/6. doi: 10.1049/ic:19971322

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Petr Blažek

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cieľom tejto práce je realizovanie pracoviska, ktoré bude simulovať sieťovú infraštruktúru priemyselných automatizačných protokolov. V tejto práci sa zameriavame a vykonávame analýzu priemyselných protokolov ModBus, Profibus, Profinet, EtherCAT, CANopen. Ďalej bude sprevádzkovaná sieťová infraštruktúra s PLC s vybranými protokoly. Výstupom práce bude pracovisko, ktoré bude schopné simulovať reálnu komunikáciu dvoch vybraných protokolov medzi dvoma a viacerými zariadeniami. Ďalšou časťou práce bude simulácia troch typov zariadení vyskytujúcich sa v priemyselnom prostredí.

KĽÚČOVÉ SLOVÁ

ModBus, Profibus / Profinet, EtherCAT, CANopen.

ABSTRACT

The aim of this work is to implement a workplace that will simulate the network infrastructure of industrial automation protocols. In this work we focus on and perform analysis of industrial protocols ModBus, Profibus, Profinet, EtherCAT, CANopen. Furthermore, the network infrastructure with PLC with selected protocols will be operated. The output of the work will be a workplace that will be able to simulate real communication of two selected protocols between two and more devices. Another part of the work will be the simulation of three types of equipment occurring in an industrial environment.

KEYWORDS

ModBus, Profibus / Profinet, EtherCAT, CANopen.

SUROVÝ, Roman. *Simulace průmyslových automatizačních protokolů*. Brno, Rok, 70 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Petr Blažek

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Simulace průmyslových automatizačních protokolů“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Petr Blažekovi, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Priemyselné komunikačné zbernice sú v anglosaskej literatúre označované pod pojmom „fieldbus“ tento protokol, ktorý bol odvodený z nemeckého „feldbus“ voľne preložené prevádzková zbernica sa používa na spodných úrovniach automatizovaného výrobného systému.

V tejto práci sa budeme zameriavať na priemyselné automatizačné protokoly a budeme ich jednotlivo analyzovať, ďalej realizovať pracovisko, ktoré bude simulovať sieťovú infraštruktúru. Poukážeme na základné a dôležité časti, ktoré jednotlivé protokoly majú a na ktoré sa rozdeľujú, čo s nimi môžeme robiť, aké sú ich hlavné vlastnosti a poukážeme si aj samozrejme na tie zlé ak existujú, čo by to pre nás znamenalo a ako by sme sa im mohli poprípade vyhnúť, alebo ich odstrániť ak je to možné. V ďalšiom rade prečo sú dôležitou súčasťou nášho života a na čo ich vlastne vieme používať. Ako môžeme naše dáta chrániť a tým spojené súbory, ktorých je každodenne na internete viac a viac, súčastne samozrejme aj viac hrozieb. Mali by sme rozumieť akými spôsobmi sa v priemyselných automatizačných a počítačových sieťach pracuje. Čo sa týka priemyselných automatizačných protokolov, sú neoddeliteľnou súčasťou nášho sveta pretože ich použitie je dvoležité s hľadiska priemyselnej komunikácie. Bez nich by sme neboli schopní prenášať súbory a samozrejme rôzne dáta.

V prvej kapitole tejto diplomovej práce sa zaoberáme automatizačnými protokolmi ako sú ModBus, Profibus/Profinet, EtherCAT a CANopen. Prvý menovaný ModBus je protokol sériovej komunikácie nazývaný bol vyvinutý spoločnosťou Modicon. Druhým z rady je Profibus je to inteligentná technológia priemyselnej zbernice a zariadenia v systéme sa na ňu pripájajú k centrálnej linke a po pripojení tieto zariadenia dokážu efektívne komunikovať a predávať informácie, ale môžu ísť aj nad rámec automatizačných správ. Ďalším v poradí často používaným je Profinet, je moderný koncept distribuovaných automatizačných štandardov. Je založený na ethernete a integruje existujúce systémy priemyselnej zbernice a to najmä Profibus, jednoducho a bez zmien. Ďalším protokolom ktorý bude spomínaný v tejto práci je CANopen. Jednou z charakteristických vlastností protokolu CANopen je podpora výmeny údajov na rôznych úrovniach zložitosti, napr. na úrovni ovládania, ako aj pomocou jednoduchých snímačov a akčných členov. Tým sa predovšetkým zabráni používaniu brán a mostov, čo zvýši náklady na implementáciu. CANopen je sieťový systém založený na sériovej zbernici CAN. Posledným a významným protokom je EtherCAT bol vyvinutý spoločnosťou Beckhoff a je založený na protokole CANopen a na Ethernete, ale líši sa od internetovej komunikácie alebo sieťovej komunikácie tým, že je špeciálne optimalizovaný na riadenie priemyselnej automatizácie.

V druhej kapitole vyberieme dva so spomenutých protokolov, ktoré konkrétne

otestujeme a budeme simulovať reálnu komunikáciu dvoch vybraných protokolov. V rámci tohoto projektu sa zameriame na analýzu a vybrané protokoly budeme postupne môcť realizovať, porovnávať ich klady a zápory. Keďže budú dva budeme môcť porovnať ich klady a zápory, ktoré bude výhodnejšie použiť a v akých okolnostiach je výhodné dané protokoly použiť.

1 Priemyselné automatizačné protokoly

Existuje veľké množstvo zbernicových protokolov v priemysle, používaných v rámci automatizácie strojov. Siete priemyselného ethernetu nám umožnujú prepájať nové, ale aj existujúce systémy a poskytovať tak predikovateľný výkon a tento si okrem iného aj udržiavajú. Okrem kompatibility, ktorá je zásadnou súčasťou protokolov je potrebné myslieť aj na fyzickú vrstvu. Na spodnej vrstve musia zasa systémy priemyselného ethernetu používané v priemyselnej praxi poskytovať prepojiteľnosť a to aj na vyšších vrstvách v rámci modelu OSI. Ako bolo spomenuté v úvode v prvej kapitole tejto diplomovej práce sa budeme zaoberať automatizačnými protokolmi ako sú ModBus, Profibus/Profinet, EtherCAT a CANopen.

1.1 Modbus

1.1.1 História

Protokol sériovej komunikácie nazývaný Modbus bol vyvinutý spoločnosťou Modicon a neskôr publikovaný spoločnosťou v roku 1979 pre použitie s jeho programovateľnými logickými radičmi (PLC) PLC je vysvedlené v predošlej kapitole. Prvé začiatky spoločnosti Modicon siahajú do roku 1968 s základnou skupinou inžinierov pod vedením Dicka Morleyho. Spoločnotosť Modicon ktorá vlastne vynašla prvý programovateľný logický ovládač dokumentáciu možno nájsť na [1].

ModBus je otvorený štandard čo znamená, že výrobcovia ho môžu zabudovať do svojich zariadení bez toho, aby museli platiť licenčné poplatky čo je jeho veľkou výhodu. Je to najrozšírenejší komunikačný protokol v priemyselnej automatizácii a je najbežnejšie dostupným spôsobom pripojenia priemyselných elektronických zariadení.

Spoločnosť Modbus predstavila koncept údajov na výrobnej hale. Umožnil pripojenie celej skupiny zariadení pomocou iba dvoch vodičov na ovládači. To samo o sebe ušetrilo obrovské investície do drôtov, práce a času na inštalácie. Namiesto kilometrov drôtov spájajúcich stovky zariadení je možné prepájať jednoduchým dvojvodičovým párom jedno reťaze z jedného zariadenia na druhé. Pre svoju dobu to bolo revolučné. ModBus si tak našiel cestu do veľkého množstva zariadení, ktoré dnes nájdeme vo všetkom od regulátorov ventilov, cez motorové pohony, HMI, až po systémy filtrácie vody a veľa ďalších. Skutočne by bolo ťažké pomenovať kategóriu produktu v priemyselnej, alebo stavebnej automatizácii, ktorá nepoužíva Modbus.

1.1.2 Prehľad ModBus

Komunikačný protokol ModBus je starým sieťovým odvetvím. ModBus prešiel skúškou času a stále sa používa vširokej škále aplikácií vrátane priemyselnej automatizácie, riadenia procesov, automatizácie budov, dopravy, energie a diaľkového monitorovania.

Je možné nájsť prakticky akýkoľvek typ senzorových a kontrolných zariadení, ktoré zahŕňajú sieť ModBus, vrátane programovateľných logických automatov (PLC), procesných regulátorov, procesných prístrojov, procesných senzorov, PID regulátorov, motorových pohonov, meračov energie, dohľadu a zberu dát (SCADA). systémy, programovateľné automatizačné ovládače (PAC), diskrétne senzory, ventily a mnoho ďalších zabudovaných zariadení dokumentáciu možno nájsť na [2].

Jednoducho povedané, Modbus je metóda používaná na prenos údajov cez sériové linky medzi elektronickými zariadeniami. Pôvodne určený na komunikáciu medzi programovateľnými logickými radičmi (PLC) a počítačmi, sa stal de facto štandardným komunikačným protokolom na pripojenie širokej škály priemyselných elektronických zariadení.

Modbus je mimoriadne kompaktný a flexibilný protokol, ktorý stále dokazuje, že sa dá prispôsobiť na použitie v širokej škále aplikácií a médií. Je obľúbený pre vzdialené aplikácie, ktoré komunikujú takmer akýmkoľvek spôsobom, vrátane káblových a mobilných telefónov, licencovaných a nelicencovaných rádií a satelitov.

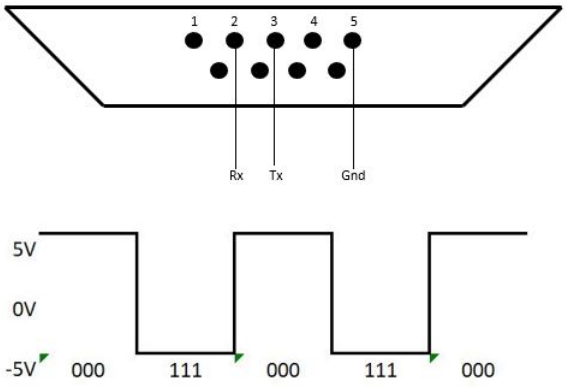
1.1.3 Načo sa to používa ModBus

Stal sa tak štandardným komunikačným protokolom v priemysle a teraz je najbežnejšie dostupným prostriedkom na pripojenie priemyselných elektronických zariadení čo ho vlastne robí využitelným. Je široko používaný mnohými výrobcami v mnohých priemyselných odvetviach. Modbus sa zvyčajne používa na prenos signálov z prístrojov a riadiacich zariadení späť do hlavného kontroléra, alebo systému zberu údajov čo si neskôr ukážeme na našom príklade ďalej napríklad do systému, ktorý meria teplotu a vlhkosť a oznamuje výsledky počítaču. Modbus sa často používa na spojenie kontrolného počítača so vzdialenou terminálovou jednotkou (RTU) v systémoch dohľadu a získavania údajov (SCADA). Verzie protokolu Modbus existujú pre sériové linky (Modbus RTU a Modbus ASCII) a pre Ethernet (Modbus TCP) dokumentáciu možno nájsť na [3].

1.1.4 Ako to funguje

Modbus sa prenáša cez sériové linky medzi zariadeniami. Najjednoduchším nastavením by mal byť jediný sériový kábel obr. 1.1 spájajúci sériové porty na dvoch

zariadeniach, Master a Slave. Dáta sa odosielajú ako série jednotiek a núl, ktoré sa nazývajú bity. Každý bit sa vysiela pomocou napätia čiže sa jedná o zmenu napätí. Nuly sa vysielajú ako kladné napätia a jednotky ako záporné. Typická prenosová rýchlosť je 9600 bitov za sekundu dokumentáciu možno nájsť na [2].



Obr. 1.1: Iterpretácia kábla

1.1.5 Kódovanie údajov

ModBus používa reprezentáciu „big-endian“ pre adresy a dátové položky. Princíp „big-endian“ uloží na pamäťové miesto s najnižšou adresou najvýznamnejší bajt (MSB) a zaň sa ukladajú ostatné bajty až po najmenej významný bajt (LSB) na konci. To v praxi znamená, že keď sa prenáša numerická veľkosť väčšia ako jeden bajt, najskôr sa odošle najvýznamnejší bajt. To môže vyzerať takto ako obr. 1.2.

16 – bits	0x1234	môže byť	0x12	0x34		
32 – bits	0x12345678L	môže byť	0x12	0x34	0x56	0x78

Obr. 1.2: Kódovanie údajov

1.1.6 Interpretácia referenčných čísel

ModBus zakladá svoj dátový model na rade tabuliek, ktoré majú rozlišovacie charakteristiky. Tu sú štyri hlavné prenášaných dát obr. 1.3 Rozlišovanie medzi vstupmi a

výstupmi a medzi bitovými adresami a údajmi adresovateľnými slovami neznamená žiadne aplikačné správanie. Je úplne prijateľné a veľmi bežné považovať všetky štyri tabuľky za vzájomne sa prekrývajúce, a je to najprirodzenejší výklad predmetného cieľového stroja dokumentáciu možno nájsť na [4].

Označenie	Význam
input discretes	Jeden bit určený iba k čítaniu príklad binární vstup
output discretes	Jeden bit, meniteľný aplikačným programom, čítanie a zápis
input registers	16-bitové množstvo poskytované systémom I / O, len na čítanie
output registers	16-bitové množstvo poskytované systémom I / O, len na čítanie

Obr. 1.3: Interpretácia referencie

Protokol tiež umožňuje pre každú z primárnych tabuliek individuálny výber čo má za nasledok údajové položky a operácie čítania alebo zápis týchto položiek sú navrhnuté tak, aby zahŕňali viacero po sebe idúcich údajových položiek až do limitu veľkosti údajov, ktorý je závislý od kódu funkcie transakcie. Neexistuje žiadny predpoklad, že údajové položky predstavujú skutočné súvislé pole údajov, hoci to je interpretácia, ktorú používajú najjednoduchšie PLC. Softvér Simply Modbus štandardne používa adresovanie s 1 bajtom. Pretože sa na definovanie adresy slave obvykle používa jediný bajt a každý slave v sieti vyžaduje jedinečnú adresu, počet slave v sieti je obmedzený na 256. Limit definovaný v špecifikácii modbus je ešte nižší na 247. Po zadaní adresy vyššej ako 255 sa softvér automaticky prepne na 2-bajtové adresovanie a zostane v tomto režime pre všetky adresy, kým sa manuálne nevypne 2-bajtové adresovanie. Druhý bajt poslaný Masterom je funkčný kód. Toto číslo hovorí slave, čiže oponentovi, ku ktorej tabuľke sa má pristupovať a či má čítať alebo zapisovať do tabuľky. Tabuľka vyzerá takto obr. 1.4 je to vlastne kód funkcie dokumentáciu možno nájsť na [4].

1.2 Profibus

1.2.1 História

Profibus sa vyvíjal na konci osemdesiatich rokov kombinovaným tlakom nemeckej vlády a nemeckých spoločností, ďalších vedúcich predstaviteľov priemyslu. Vznikala potreba riešenia automatizácie. Ich snaha tak vytvorila automatizačné riešenie, ktoré je nielen dnes životaschopné, ale viedlo aj k ďalším riešeniam, ktoré si postupne

Funkčný kód	Akcia	Tabuľkové meno
01 (01 hex)	Čítanie	Discrete Output Coils
05 (05 hex)	Zápis jedného	Discrete Output Coil
15 (0F hex)	Zápis viacerých	Discrete Output Coils
02 (02 hex)	Čítanie	Discrete Input Contacts
04 (04 hex)	Čítanie	Analog Input Registers
03 (03 hex)	Čítanie	Analog Output Holding Registers
06 (06 hex)	Použitie jedného	Analog Output Holding Register
16 (10 hex)	Použitie Viacerých	Analog Output Holding Registers

Obr. 1.4: Kód funkcie

prejdeme v tejto práci. Hrdé dedičstvo Profibus umožňuje mnohým európskym zákazníkom obrátiť sa na automatizáciu špecifickú pre ich potreby. Ústredná asociácia pre elektrotechnický priemysel vytvorila Profibus. V roku 1987 sa 21 spoločností a inštitúcií v Nemecku spojilo a vytvorilo nový protokol. Ich cieľom bolo vytvoriť bit-sériový systém Fieldbus, aby bol systém životaschopný, potrebovali štandardizovať rozhranie zariadenia poľa. Skupina, ktorá získala názov Centrálna asociácia pre elektrotechnický priemysel (ZVEI), dokončila svoj cieľ a vytvorila Profibus FMS (Fieldbus Message Specification). Profibus bol prvýkrát propagovaný v roku 1989.

Tento nový protokol uspokojil štandardizáciu priemyselnej automatizácie ktorá sa mohla ďalej rozvíjať prostredníctvom protokolu schopného odosielať komplexnú komunikáciu. ZVEI však ešte nebol plne dokončený. V roku 1993 skupina predstavila nový štandard, Profibus DP (Decentralized Periphery). Tento nový protokol bol jednoduchší, ľahšie konfigurovateľný a umožnoval rýchlejšie zasielanie správ. ZVEI naďalej slúži v elektronickom priemysle v Nemecku. Ich práca na vytvorení Profibus bola nevyhnutná dokumentáciu možno nájsť na [10].

1.2.2 Organizácie

Normy Profibus sa udržiavajú, aby nezaostávali pozadu a rozširujú prostredníctvom niekoľkých dôležitých organizácií, ktoré sa postupom času začali vytvárať. V roku 1989 výrobcovia a používatelia Profibus vytvorili Organizáciu používateľov Profibus (PNO). Táto skupina bola a stále je nekomerčným podnikom. Členovia sa usilujú o pokrok v systéme Profibus prostredníctvom podpory a vzdelávania vrátane publikovania dokumentov, ktoré používateľom pomáhajú uspokojiť ich potreby pomocou existujúcich technológií dokumentáciu možno nájsť na [6].

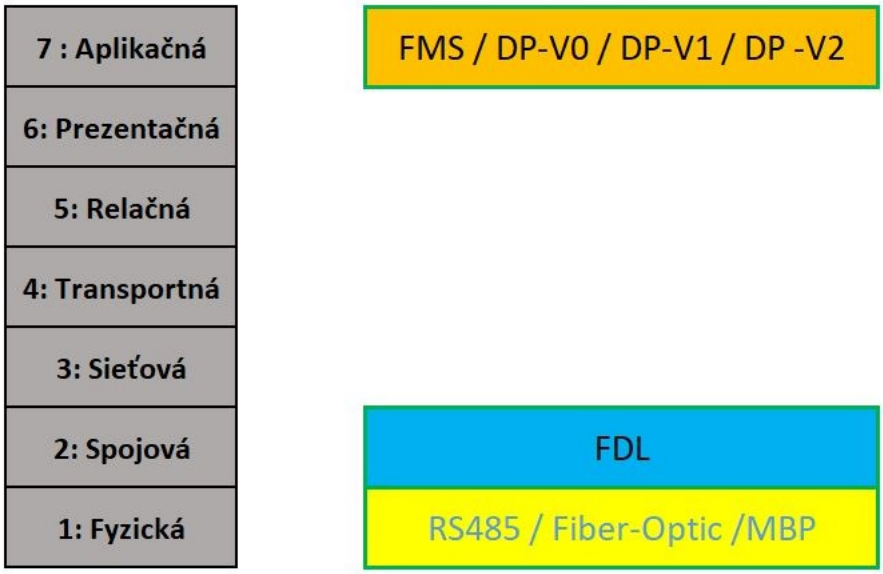
1.2.3 Prehľad Profibus

Profibus je inteligentná technológia priemyselnej zbernice, zariadenia v systéme sa pripájajú k centrálnej linke a po pripojení tieto zariadenia dokážu efektívne komunikovať, a môžu ísť aj nad rámec automatizačných správ. Profibus je 2-vodičový priemyselný dátový komunikačný štandard, ktorý umožňuje komponentom, ako sú snímače, akčné členy a ovládače, vymieňať si procesné hodnoty a dosiahnuť automatizáciu celého procesu. V automatizačnej technike ako sme vyššie spomínali prvýkrát bol propagovaný v roku 1989. Je to najbežnejšie používaná priemyselná zbernica s viac ako 50 miliónmi nainštalovaných zariadení. Zariadenia Profibus sa môžu podieľať aj na vlastnej diagnostike a diagnostike pripojenia. Profibus nie je jediný komunikačný systém, ale celý rad protokolov postavených na rovnakom zväzku technológie Fieldbus. Užívatelia môžu kombinovať rôzne protokoly Profibus s vlastným softvérom a ďalšími požiadavkami, čo vedie k jedinečnému aplikačnému profilu. Vďaka mnohým dostupným profilom môže Profibus vyhovovať špecifickým potrebám. Jedna vec však zostáva rovnaká. Vďaka dôkladnému testovaniu spĺňajú zariadenia Profibus vysoký štandard kvality, ktorý sa hodí do siete vysokej kvality. Na najzákladnejšej úrovni Profibus ťaží z vynikajúcej konštrukcie svojich vrstiev OSI a základnej topológie dokumentáciu možno nájsť na [7].

1.2.4 OSI model

Siete Profibus využívajú tri samostatné vrstvy sieťového modelu OSI. Najskôr Profibus popisuje aplikačnú vrstvu. Avšak existuje aj niekoľko iných verzií Profibus, ktoré spracovávajú rôzne typy správ na aplikačnej vrstve. Niektoré typy správ, ktoré podporuje Profibus, zahŕňajú cyklickú a acyklickú výmenu údajov, diagnostiku, spracovanie alarmov a izochrónne správy. Profibus OSI Model možno vidieť na obr. 1.5 Profibus nám nedefinuje tri vrstvy, ale až šesť. Definuje tak dátové spojenie a fyzické vrstvy, vrstvy jedna a dve. Vrstva dátového spojenia je dokončená prostredníctvom dátového spojenia Fieldbus alebo FDL. Systém FDL kombinuje dve spoločné schémy, metodiku master-slave a odovzdávanie tokenov. V sieti typu master-slave masters, zvyčajne radiče nám posielajú žiadosti slaves, senzorom a ovládačom. Slaves zodpovedajúco Profibus tiež obsahuje odovzdávanie tokenov a systém v ktorom sa medzi uzlami prenáša signál „token“. Komunikovať môže iba uzol s tokenom.

Nakoniec Profibus definuje fyzickú vrstvu, ale ponecháva si priestor na flexibilitu čo nám z neho robí systém ktorý je adaptívny. Systémy Profibus môžu mať tri typy médií. Prvým je štandardný elektroinštalačný systém s krútenými pármi, v tomto prípade RS485. K dispozícii sú aj dva pokročilejšie systémy. Systémy Profibus môžu



Obr. 1.5: Profibus OSI Model

teraz fungovať pomocou optického prenosu v prípadoch, keď je to vhodnejšie samozrejme táto varianta je pre zákazníka pomerne drahšia. Bezpečnostne vylepšený systém s názvom Manchester Bus Power, ďalej MBP je k dispozícii aj v pomienkach ako je chemické prostredie, alebo prostredie náchylné na výbuch dokumentáciu možno nájsť na [7].

1.2.5 Topológia

Profibus využíva topológiu zbernice. V tejto topológii je v systéme zapojená centrálna linka alebo BUS. Zariadenia sú pripojené k tejto centrálnej zbernici. Jedna zbernica eliminuje potrebu vedenia po celej dĺžke vedúceho od centrálneho ovládača k jednotlivým zariadeniam.

V minulosti sa každé zariadenie Profibus muselo pripájať priamo k centrálnej zbernici. Bolo spomenuté, že došlo k určitému technologickému pokroku a to nám umožnil nový „dvojvodičový“ systém. V tejto topológii sa centrálna zbernica Profibus môže pripojiť k systému Ethernet ProfiNet o ktorom budeme hovoriť neskôr. Týmto spôsobom je možné navzájom prepojiť viacero zberníc Profibus dokumentáciu možno nájsť na [10].

1.2.6 Typy Profibus

Postupom času Profibus postúpil niekoľko revízií v podstate sa vyvíjal. V niektorých prípadoch tieto pokroky viedli k novému typu zbernice Profibus. V iných prípadoch znamenajú nové revízie rôzne verzie toho istého typu Profibus. Rôzne riešenia Profibus v každom prípade znamenajú, že systém môže byť prispôsobený rôznym potrebám rôznych odvetví.

- **Profibus FMS**

  Počiatočná verzia systému Profibus bola Profibus FMS, špecifikácia správy priemyselnej zbernice. Profibus FMS bol navrhnutý tak, aby komunikoval medzi programovateľnými ovládačmi a počítačmi a medzi nimi posielal komplexné informácie. Bohužiaľ, ako počiatočné úsilie dizajnérov Profibus, technológia FMS nebola tak flexibilná, ako bolo potrebné. Tento protokol nebol vhodný pre menej zložité správy alebo komunikáciu v širšej a komplikovanejšej sieti. Nové typy Profibus by uspokojili tieto potreby. PROFIBUS FMS sa stále používa, aj keď prevažná väčšina používateľov považuje novšie riešenia za vhodnejšie.

- **Profibus DP**

  Druhý typ zbernice Profibus je univerzálnejší. Tento nový protokol s názvom ProfibusDP pre decentralizované periférie je omnoho jednoduchší a rýchlejší. PROFIBUS DP sa používa v drvivej väčšine aplikačných profilov Profibus, ktoré sa dnes používajú. Profily aplikácií umožňujú používateľom kombinovať ich požiadavky na konkrétne riešenie a čoskoro sa o nich bude diskutovať podrobnejšie. Samotný Profibus DP má tri samostatné verzie. Každá verzia DP-V0, DP-V1 a DP-V2, poskytuje novšie a zložitejšie funkcie:
- DP-V0 je určený na výmenu cyklických dát / diagnostiky
- DP-V1 je určený na acyklickú / cyklickú výmenu dát a spracovanie alarmov
- DP-V2 je určený na izochrónnu výmenu dát a komunikáciu medzi podriadenými stanicami.

- **Profibus PA**

  Profibus PA je protokol určený pre automatizáciu procesov. V skutočnosti je Profibus PA typom profilu aplikácie Profibus DP. Profibus PA štandardizuje proces prenosu nameraných údajov. Má však veľmi dôležitú jedinečnú vlastnosť. Profibus PA bol navrhnutý špeciálne pre použitie v nebezpečnom prostredí.

1.2.7 Technológia MBP

Médium MBP bolo navrhnuté špeciálne na použitie v Profibus PA. Umožňuje prenos údajov aj energie čo je jeho veľká výhoda, avšak táto technológia znižuje výkon. Menší výkon nám až tak nevadí keďže sa takmer vylučuje možnosť výbuchu. BUS využívajúce MBP môžu dosiahnuť prenos na 1900 metrov a môžu podporovať pobočky a profily aplikácií.

Profibus môže byť prispôsobený konkrétnym potrebám pomocou aplikačných profilov. Existuje mnoho profilov, ktoré kombinujú štandardy pre prenosové médiá, komunikačný protokol (FMS, DP-V0 atď.) A jedinečné protokoly. Každý profil aplikácie je prispôsobený konkrétnemu použitiu a pravidelne sa objavujú nové profily. Ich zoznam by bol ťažkopádny. Niektoré profily aplikácií sú však rozšírené. Dva príklady sú PROFIsafe a PROFIdrive.

- **PROFIsafe**

  PROFIsafe používa ďalší softvér na vytvorenie vysoko integrovanej siete. Táto sieť je užitočná v situáciách, keď sa vyžaduje vysoká bezpečnosť, aby boli dodávatelia a výrobcovia certifikovaní v PROFIsafe, musia si udržiavať vysoké štandardy kvality dokumentáciu možno nájsť na [9].

- **PROFIdrive**

  PROFIdrive bol vytvorený pre aplikácie na riadenie pohybu. Softvér pridaný do špecifikácie PROFIBUS DP umožňuje sieti dosiahnuť presnú kontrolu servomotorov a ďalších zariadení. PROFIdrive tak môže dosiahnuť synchronizáciu v celej sieti dokumentáciu možno nájsť na [9].

1.2.8 Quality Assurance

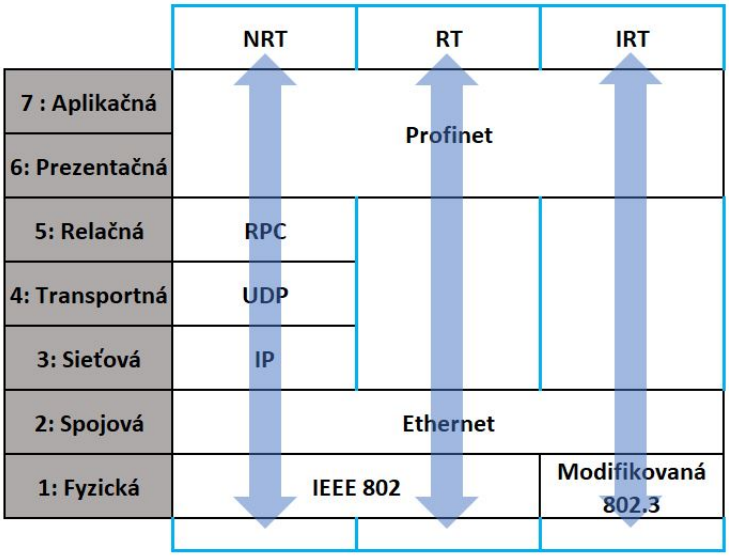
Organizácia používateľov Profibus vytvorila program testovania zhody s cieľom zabezpečiť, aby zariadenia spĺňali vysoké štandardy. V tomto programe sa zariadenie odošle do nezávislého laboratória na testovanie. Zariadenie potom absolvuje komplexnú sériu testov, medzi inými aj testy hardvéru, zhody a funkcie. Výsledky testov sú zdokumentované dokumentáciu možno nájsť na [7].

Keď zariadenie prejde všetkými skúškami, jeho výrobca môže požiadať o osvedčenie o zhode. Certifikát je platný tri roky a je možné ho obnoviť ďalším testovaním.

1.3 Profinet

1.3.1 Úvod do Profinetu

Profinet je moderný koncept distribuovaných automatizačných štandardov. Je založený na ethernete a integruje existujúce systémy priemyselnej zbernice a to najmä Profibus, jednoducho a bez zmien. Toto je veľmi dôležitý aspekt pre splnenie požiadavky na konzistentnosť od úrovne podnikového riadenia po úroveň v teréne. Ďalej predstavuje kľúčový príspevok k zabezpečeniu bezpečnosti používateľa pre jeho investíciu do toho, že existujúce časti systému môžu byť začlenené do Profinet-u bez toho, aby ich bolo potrebné meniť. Profinet je najvyspelejšie riešenie pre priemyselný Ethernet na svete. Je to komunikačný protokol na výmenu údajov medzi radičmi a zariadeniami. Ovládače môžu byť PLC, DCS alebo PAC dokumentáciu možno nájsť na [11].



Obr. 1.6: Ako sa spolieha Profinet na štandardné protokoly na prenos

Zariadenia môžu byť vstupno-výstupné bloky, systémy videnia, RFID čítačky, disky, procesné nástroje, servery proxy alebo dokonca ďalšie kontroléry. Aby sme pochopili, ako Profinet presúva informácie, pomáha mať na pamäti, aké informácie sa pohybujú. Musia sa informácie doručiť okamžite? Je to kritické z hľadiska bezpečnosti? Je to veľké množstvo informácií, ktoré sa odosielajú iba raz. Tieto rôzne typy informácií si vyžadujú rôzne prenosové mechanizmy a tieto mechanizmy tvoria „komunikačné kanály“. V Profinet-e sú tri komunikačné kanály: Real-Time (RT),

Non-Real-Time (NRT) a Isochronous Real-Time (IRT) dokumentáciu možno nájsť na [14].

Tieto komunikačné kanály sa pri výmene údajov v sieti Ethernet spoliehajú na množstvo sieťových protokolov. Namiesto zlúčenia všetkých týchto protokolov do jedného košíka sa ich pokúsime usporiadať do rámca, ktorý ich bude sledovať. Tento rámec sa nazýva model OSI (Open Systems Interconnection). O modeli OSI je možné vyhladať na internete veľa použiteľních zrojov. Dokumentáciu možno nájsť na [11].

Tu je niekoľko ďalších výhod ktoré ponúka Profinet na úrovni IO:

- Vysoko škálovateľné architektúry
- Prístup k poľným zariadeniam prostredníctvom siete
- Údržba a servis odkiaľkoľvek (aj cez internet)
- Najlepšie vo svojej diagnostike vo svojej triede
- Nižšie náklady na monitorovanie údajov o výrobe / kvalite

Profinet vyhovuje požiadavkám automatizácie IT a výrobného procesu pomocou troch komunikačných kanálov:

- Štandardný TCP / IP
- Reálny čas (Profinet RT)
- Isochronous Real Time

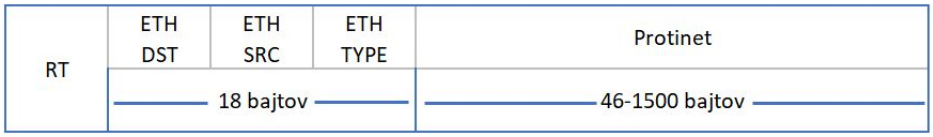
Profinet sa spolieha na mnoho štandardných protokolov na prenos dát v sieťach. Dokumentáciu možno nájsť na [**?**] A pretože tieto protokoly majú relatívne silné a slabé stránky, Profinet ich používa iba vtedy, keď to absolútne musí. Vidno to na obr. 1.6.

1.3.2 Kanál v reálnom čase (RT)

Latencia a jitter sú zlé správy pre protokol priemyselnej automatizácie „v reálnom čase“. Preto Profinet navrhol kanál „Real Time“, aby sa pokúsil znížiť obidve tieto hodnoty. Kanál RT preskočí kroky zapuzdrenia vo vrstvách Network, Transport and Session. To znamená, že rámce vymieňané cez kanál RT majú nízku latenciu a nízky jitter, ale existuje tu aj skutočná nevýhoda: neexistuje IP adresa. A to znamená, že RT snímky sa nedajú smerovať medzi LAN. Profinet používa na prenos údajov v reálnom čase iba jednu a dve vrstvy OSI vidno to na obr. 1.7. Jeden kábel lepšie povedané drôt a to znamená, že v rámci sú spolu s Ethertype zabalené iba cieľové a zdrojové adresy MAC dokumentáciu možno nájsť na [13].

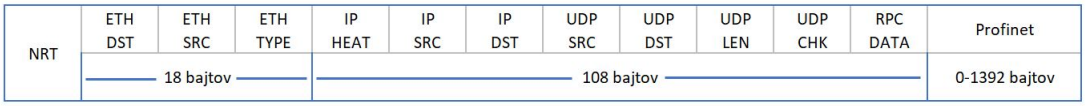
1.3.3 Kanál bez reálneho času (NRT)

Obmedzenia smerovania môžu byť skutočným problémom vo veľkých sieťach, kde diagnostické nástroje musia mať prístup k zariadeniam PROFINET, aby mohli udr-



Obr. 1.7: Kanál v reálnom čase

žiavať prehľad o prevádzkovom stave siete Profinet. Profinet má tiež komunikačný kanál „Non Real Time“ (NRT). Používa tak všetky vrstvy modelu OSI a má adresy IP. Sledovaný Profinet tak môže pristupovať k zariadeniam z rôznych smerovacích hraníc alebo dokonca z Internetu. Kompromisom je však vyššia latencia a jitter pre tieto NRT komunikácie. Profinet používa kanál NRT na menej časovo citlivú komunikáciu, napríklad na vytvorenie spojenia medzi zariadením a radičom alebo na prístup k diagnostickým údajom z externej siete. Táto zvýšená flexibilita však prináša zvýšenú réžiu údajov - 108 bajtov navyše na každom pakete vidno to na obr. 1.8.



Obr. 1.8: Kanál v reálnom čase

1.3.4 Izolačný kanál v reálnom čase (IRT)

Rovnako ako je to v prípade RT kanála, má stále nevyhnutný jitter z prenosu cez štandardné ethernetové prepínače. Prepínače môžu fungovať ako dopravné križovatky a môžu zúžiť viacero tokov údajov na jedno pripojenie. A rovnako ako preplnená križovatka, prepínače môžu pridať neočakávané oneskorenia v premávke.

IRT eliminuje tieto oneskorenia pridaním a to tým, že k pravidlám používaným na prepínanie ethernetového prenosu a vytváraním špeciálnych pravidiel pre prenos Profinet. Pridáva však aj niektoré rozšírenia do bežného IEEE 802.3 Ethernet na implementáciu niečoho ako „HOV Lane“ pre IRT prenos. Technicky premieňa stochastickú CSMA-CD (Multiple Access Access Sense - Collision Detection) sieť na deterministickú TDMA (Time Division Multiple Access) sieť, ale táto úroveň detailov je najlepšie ponechať článku o IRT dokumentáciu možno nájsť na [13].

1.4 CANopen

1.4.1 Historia o CANopen

Vo februári 1986 predstavil Robert Bosch na kongrese SAE v Detroite systém sériovej zbernice CAN. Bol navrhnutý tak, aby spracovával krátke správy (až 8 bajtov), podporoval prístup typu master (čo znamená, že kolízie sa riešia podľa priority) a ponúkalo to Vysoký stupeň spoľahlivosti (15-bitový CRC pre každú správu) V polovici roku 1987 spoločnosť Intel dodala prvý čip CAN, 82526. Krátko nato spoločnosť Philips Semiconductors uviedla radič 82C200 CAN. Dnes približne 20 výrobcov čipov vyrába zariadenia s CAN Rozhrania a takmer každý nový osobný automobil vyrobený v Európe je vybavený najmenej jednou sieťou CAN. CAN, ktorý sa používa aj v iných typoch vozidiel, od vlakov po lode, ako aj v priemyselnej kontrole, je jedným z najvýznamnejších BUS protokolov. Len v roku 1999 sa do aplikácií dostalo takmer 60 miliónov radičov CAN, v roku 2000 sa predalo viac ako 100 miliónov zariadení CAN.

Od roku 1993 vyvíjalo európske konzorcium vedené spoločnosťou Bosch v rámci projektu Esprit ASPIC prototyp prototypu toho, čo by sa malo stať CANopenom, profilom CAL pre interné vytváranie sietí výrobných buniek. V roku 1995 spoločnosť CiA vydala úplne revidovaný komunikačný profil CANopen. Rodina profilov CANopen tiež definuje rámec pre programovateľné systémy, ako aj rôzne profily zariadení, rozhraní a aplikácií. Toto je dôležitý dôvod, prečo sa celé priemyselné odvetvia (napr. Tlačiarenské stroje, námorné aplikácie, lekárske systémy) rozhodli používať CANopen koncom 90. rokov.

Predajcovia polovodičov, ktorí do svojich zariadení implementovali moduly CAN, sa samozrejme zameriavajú hlavne na automobilový priemysel. Od polovice 90. rokov 20. storočia zasielali spoločnosti Infineon Technologies (predtým Siemens) a Motorola veľké množstvo riadiacich jednotiek CAN európskym výrobcom osobných automobilov. Hoci protokol CAN je teraz „starý“, stále sa vylepšuje. V prípade protokolov vyššej vrstvy bolo vykonaných niekoľko vylepšení týkajúcich sa schvaľovania rôznych aplikácií dôležitých z hľadiska bezpečnosti a bezpečnosti. Spoločnosť Bosch nedávno predstavila verziu CAN-FD, flexibilnú rýchlosť prenosu dát, ktorá umožňuje použitie vyšších prenosových rýchlostí. Rozšírenia ako CAN-FD zvýšia celkovú životnosť CAN o ďalších desať až pätnásť rokov. Keď sa vezme do úvahy, že CAN je stále na začiatku prenikania na globálny trh, aj konzervatívne odhady ukazujú ďalší rast tohto BUS systému na nasledujúcich desať až pätnásť rokov dokumentáciu možno nájsť na [16].

1.4.2 Úvod do protokolu CANopen

CANopen spája automatizačné zariadenia pomocou vzájomného zasielania správ. CANopen, postavený na štandardnom štandarde fyzickej komunikácie CAN (Controller Area Network), používa hardvér CAN na definovanie protokolu aplikačnej vrstvy, ktorý štruktúruje úlohu konfigurácie, prístupu a zasielania správ medzi rôznymi druhmi automatizačných zariadení. Dátová štruktúra CANopen je tiež základom novšieho vysokorýchlostného protokolu EtherCAT dokumentáciu možno nájsť na [17].

1.4.3 Dátová štruktúra

CANopen je objektový komunikačný protokol.Zariadenia CANopen nie sú striktne zariadenia typu peer-peer a striktne zariadenia typu Master-Slave. Je možné, že zariadenie CANopen je Master pre ďalšie zariadenie CANopen a prikazuje mu vykonať nejakú akciu. Zároveň to môže byť slave zariadenie pre iné CANopen zariadenie, ktoré si želá prikázať, aby prijal nejakú akciu. A zároveň si môže vymieňať údaje typu peer s ďalším zariadením CANopen.

To je všetko možné vďaka Object Dictionary voľným prekladom Slovník objektov, organizačnú štruktúru pre všetku komunikáciu a všetky údaje vystavené sieti v zariadení CANopen. Ak je povolený prístup, zariadenie v sieti CANopen môže nakonfigurovať ďalšie zariadenie CANopen na vykonávanie vzájomnej komunikácie alebo prijímanie acyklických správ. Tento druh všeobecného prístupu sa zvyčajne nedáva k dispozícii náhodným uzlom v sieti. Namiesto toho je komunikácia vopred nakonfigurovaná a poskytnutá zariadeniam, ktoré pôsobia ako majster CANopen Master.

Object Dictionary CANopen je jadrom zariadenia CANopen. Object Dictionary CANopen obsahuje podporované typy údajov, komunikačné objekty, objekty špecifické pre dodávateľa a zariadenie a objekty špecifické pre akýkoľvek podporovaný profil dokumentáciu možno nájsť na [18].

1.4.4 CANopen Formát správy

Formát správy pre rámec CANopen je založený na formáte rámca CAN. V protokole CAN sa dáta prenášajú v rámcoch pozostávajúcich z 11-bitového alebo 29-bitového CAN-ID, riadiacich bitov, ako je napríklad vzdialený prenosový bit (RTR), počiatočný bit a pole s 4-bitovou dĺžkou dát a 0 až 8 bajtov údajov. COB-ID, bežne uvádzané v CANopene, pozostáva z CAN-ID a kontrolných bitov. V systéme CANopen je 11-bitové ID CAN rozdelené na dve časti: 4-bitový funkčný kód a 7-bitové

ID uzla CANopen. 7-bitové obmedzenie veľkosti obmedzuje počet zariadení v sieti CANopen na 127 uzlov. Na obr. 1.9 možno vidieť formát rámca CANopen.

Obr. 1.9: CANopen formát rámca (bity zobrazené okrem dátového poľa)

Všetky COB-ID musia byť jedinečné, aby sa predišlo konfliktom v zbernici. Pri komunikácii SDO by mal vždy existovať iba jeden uzol, ktorý potrebuje prístup k indexom slovníka jednotlivých objektov podriadených uzlov dokumentáciu možno nájsť na [20].

1.4.5 Objekty služieb (SDO)

Protokol CANopen tiež špecifikuje, že každý uzol v sieti musí implementovať server, ktorý spracováva požiadavky na čítanie a zápis do svojho slovníka objektov. To umožňuje, aby master servera CANopen pôsobil ako klient tohto servera. Mechanizmus priameho prístupu (čítanie / zápis) do slovníka objektov servera je objekt servisných údajov (SDO). Uzol, ku ktorému sa pristupuje do slovníka objektov, sa označuje ako server SDO a uzol, ktorý zachytáva údaje, sa nazýva klient SDO. Prenos vždy začína klient SDO.

Hlavný uzol CANopen zvyčajne pošle žiadosť do siete a požadovaný uzol odpovie na požadované údaje. CANopen používa rezervované ID správ na uľahčenie tejto komunikácie. Keď chce klient SDO požiadať zo servera informácie, pošle žiadosť SDO pomocou CAN-ID 600 h + ID uzla. Server potom odpovie pomocou CAN-ID 580 h + ID uzla. ID uzla určuje, z ktorého podriadeného uzla správa pochádza. V nižšie uvedenom príklade hlavný uzol (klient SDO) vyšle správu do siete s CAN-ID 603h. Aj keď túto správu vidia všetky uzly, všetky uzly okrem cieľového uzla ju ignorujú, pretože správa pre nich nie je určená. Cielený uzol chápe, že správa s ID 603h znamená, že správa je určená pre tento uzol, čo je požiadavka SDO. Údajové pole správy bude špecifikovať index a subindex objektu, ku ktorému by mal master získať prístup k údajom. Cielený uzol potom odpovie správou ID 583h. Dátové pole správy s odpoveďou bude obsahovať požadované údaje dokumentáciu možno nájsť na [18].

1.5 EtherCAT

1.5.1 Históia EtherCAT

Aplikačné vrstvy Ethernet, ktoré sa používajú v priemyselných a stavebných automatizačných systémoch, sú typicky aplikačné vrstvy s nízkym užitočným zaťažením. Napríklad ethernetová správa pre aplikačnú vrstvu, ako je napríklad Modbus TCP, môže niesť iba jeden register pre serverové zariadenie s malými údajmi, ako je napríklad riadiaca jednotka ventilov. V malých zariadeniach s takýmto užitočným zaťažením sa takto stratia obrovské časti pásma, pretože sa nepresúvajú nielen malé dátové pakety, ale tieto správy sa vydávajú iba vtedy, keď si klient alebo hlavné zariadenie tieto údaje vyžiadajú. Modbus TCP je mimoriadne neefektívny príklad, ale rovnaké druhy problémov so šírkou pásma a užitočným zaťažením sa nachádzajú v aplikačných vrstvách, ako sú EtherNet / IP a Profinet IO. Dokumentáciu možno nájsť na [**?**]

Nemecká automatizačná spoločnosť Beckhoff vyvinula systém priemyselnej zbernice s názvom Fast Lightbus na odstránenie problému s nízkou šírkou pásma, ktorý sa vyskytuje v iných protokoloch Ethernet. Tento protokol viedol k EtherCAT, ktorý spoločnosť Beckhoff uviedla v roku 2003.

1.5.2 Čo je EtherCAT

Ethernet je pre technológiu automatizácie riadenia EtherCAT bol vlastne vyvinutý spoločnosťou Beckhoff. Je založený na protokole CANopen a na Ethernete, ale líši sa od internetovej komunikácie alebo sieťovej komunikácie tým, že je špeciálne optimalizovaný na riadenie priemyselnej automatizácie. Normy sú definované a udržiavané technologickou skupinou EtherCAT. Dokumentáciu možno nájsť na [21]

Pri použití sieťového modelu OSI sa Ethernet a EtherCAT spoliehajú na rovnaké vrstvy fyzického a dátového spojenia. Okrem toho sa obe siete líšia svojím dizajnom, pretože sú optimalizované pre rôzne úlohy. Napríklad Ethernet je navrhnutý na odosielanie veľkého množstva dát cez mnoho rôznych uzlov. Je schopný smerovať údaje z a na miliardy samostatných adries, čo umožňuje komunikáciu v rozsiahlych sieťach.

1.5.3 Rámec EtherCAT

Na to, aby sa rámec EtherCAT mohol neustále pohybovať bez zastavenia v každom uzle, musí paket obsahovať konkrétne komponenty, podobne ako autá vlaku. Pri pohľade na rámec EtherCAT je jeho analógia s vlakom úplne zrejmá. Hlavička

funguje ako lokomotíva. Autá a ich obsah sú údaje CHOP. Pracovný pult je spolu s informáciami o preprave.

Všetky tieto časti sa jednoducho zmestia do rámu EtherCAT a rám sa zmestí jednoducho do rámu Ethernet. Ethernet je prenosové médium, ktoré umožňuje prevádzku EtherCAT. Rámec EtherCAT jednoducho nahrádza rámec IP štandardnej správy Ethernet. Rámec Ethernet preto nevyžaduje úpravy, čo opäť prispieva k flexibilite pre EtherCAT.

1.5.4 Kompatibilná rýchlosť EtherCAT

Aj keď v dátovom rámci stále existuje malé oneskorenie, keď k nemu zariadenie pridáva svoje údaje, je to výrazne znížené často jediným dátovým tokom EtherCAT v porovnaní s viacerými ethernetovými rámcami použitými v ethernetovej sieti. Táto výhoda môže byť tiež nevýhodou.Mnohé zariadenia nemusia byť schopné zvládnuť tieto veľmi skrátené časy cyklov a sieť EtherCAT bude pravdepodobne potrebné spomaliť, aby vyhovovala týmto zariadeniam. Pretože sieť EtherCAT môže byť spomalená, označte túto nevýhodu opäť za výhodu.

EtherCAT tiež využíva distribuovaný systém hodín. Táto metóda umožňuje malé chvenie bez dodatočného hardvéru a zodpovedá synchronizačnej dôležitosti požadovanej v priemyselnej automatizácii.Keď rámec EtherCAT prechádza každým uzlom, uzol pridá do svojich údajov časovú pečiatku „prijatej správy“. Každý uzol pridá časovú pečiatku pri prijatí správy a potom každý uzol znova pripojí časovú pečiatku, keď sa rámec pohybuje späť cez uzly, na ceste späť k hlavnej jednotke.Master má potom presné oneskorenie pre každý uzol, pretože dáta časovej pečiatky sa počítajú pri každom prenose dátového rámca. Dokumentáciu možno nájsť na [22]

2 Simulácia komunikácie priemyselných automatizačných protokolov

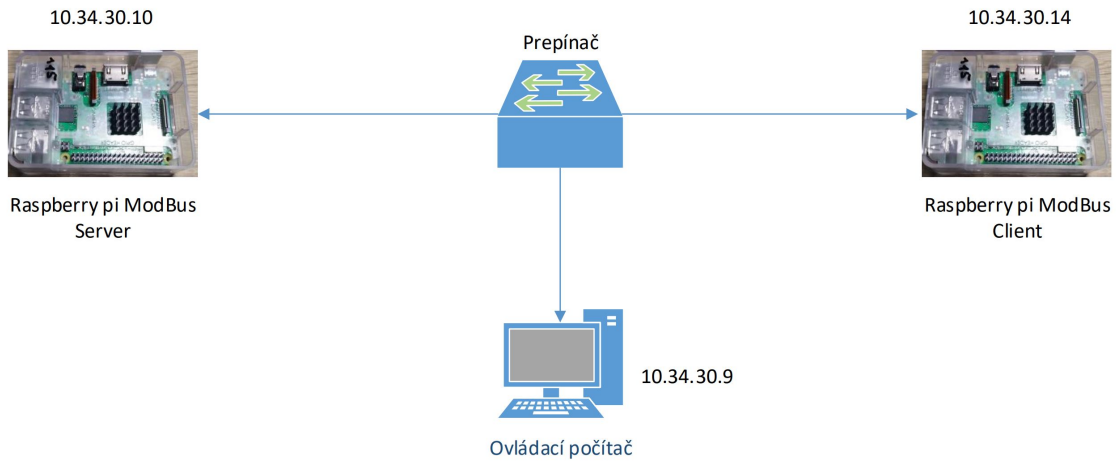
2.1 Simulácia komunikácie ModBus

Keďže výstupom tejto diplomovej práce je realizovanie pracoviska, ktoré bude simulovať sieťovú infraštruktúru priemyselných automatizačných protokolov, v rámci práce sme zvolili analýzu priemyselného prvého protokolu a tým je ModBus. Tento protokol je sériový komunikačný protokol a určený pre vzájomnú komunikáciu pre rôzne zariadenia a jeho popis sme si podrobne vysvetlili v kapitole pre ModBus. Tento protokol otestujeme medzi zariadeniami pripojenými k rovnakej sieti. Táto realizácia bola uskutočnená pomocou Raspberry Pi. Neostali sme avšak len pri tejto realizácii a odskúšali sme ho aj v reálnych podmienkach, kde je možné sledovať teplotu, tlak a ďalšie.

2.1.1 ModBus realizácia medzi dvoma Raspberry Pi

Na Raspberry Pi sme museli nainštalovať a nakonfigurovať softvér. Táto realizácia bola možná prostredníctvom PuTTY, je to vlastne klient protokolov SSH, Telnet, holého TCP a terminál pre spojenia cez sériový port. Ďalej bolo nutné vytvoriť návrh schémy zapojenia je ho možno vidieť na obr. 2.1. Pracovisko tak bolo zrealizované komunikáciou medzi ovládacím počítačom, ktorý komunikoval s dvoma Raspberry Pi prostredníctvom prepínača. Prepínač sme tam umiestnili, aby sa dalo komunikovať medzi dvoma Raspberry Pi a nebolo nutné vytvárať nové prepojenie. Jedno Raspberry pi bolo nastavené ako Server ModBus to pre náš prípad znamená, že Server bude čakať na prichádzajúce spojenie od Klienta a po nadviazaní nášho spojenia Server nám potom odpovie na otázky klienta kým, však klient pripojenie nezruší, to by malo za následok ukončenie spojenia. Tým pádom druhé Raspberry Pi je realizované ako klient ModBus.

Konfigurácia bola teda takáto prvé Raspberry Pi čiže server má IP adresu 10.34.30.10 a druhé 10.34.30.14 ovládací počítač má IP adresu 10.34.30.9. ako je vidno bolo treba nastaviť jednu podsieť čiže bolo možné komunikovať s oboma Raspberry Pi. Ďalšia konfigurácia bola pomocou jazyka Python, čo je vlastne interpretovaný, interaktívny programovací jazyk, ktorý nám pomohol s realizáciou ModBus Server a aj ModBus klient. Príklad výpisu kódu pre ModBus ServerA.1 kód je nastavený ako synchrónny server a je implementovaný v čistom pythone bez akejkoľvek tretej strany knižníc. Inicializujte tak ukladanie údajov a dátové súbory tak odpovedajú iba na adresy, na ktoré sú inicializované. Túto realizáciu môžeme vidieť na obr. 2.2 a na obr. 2.3.



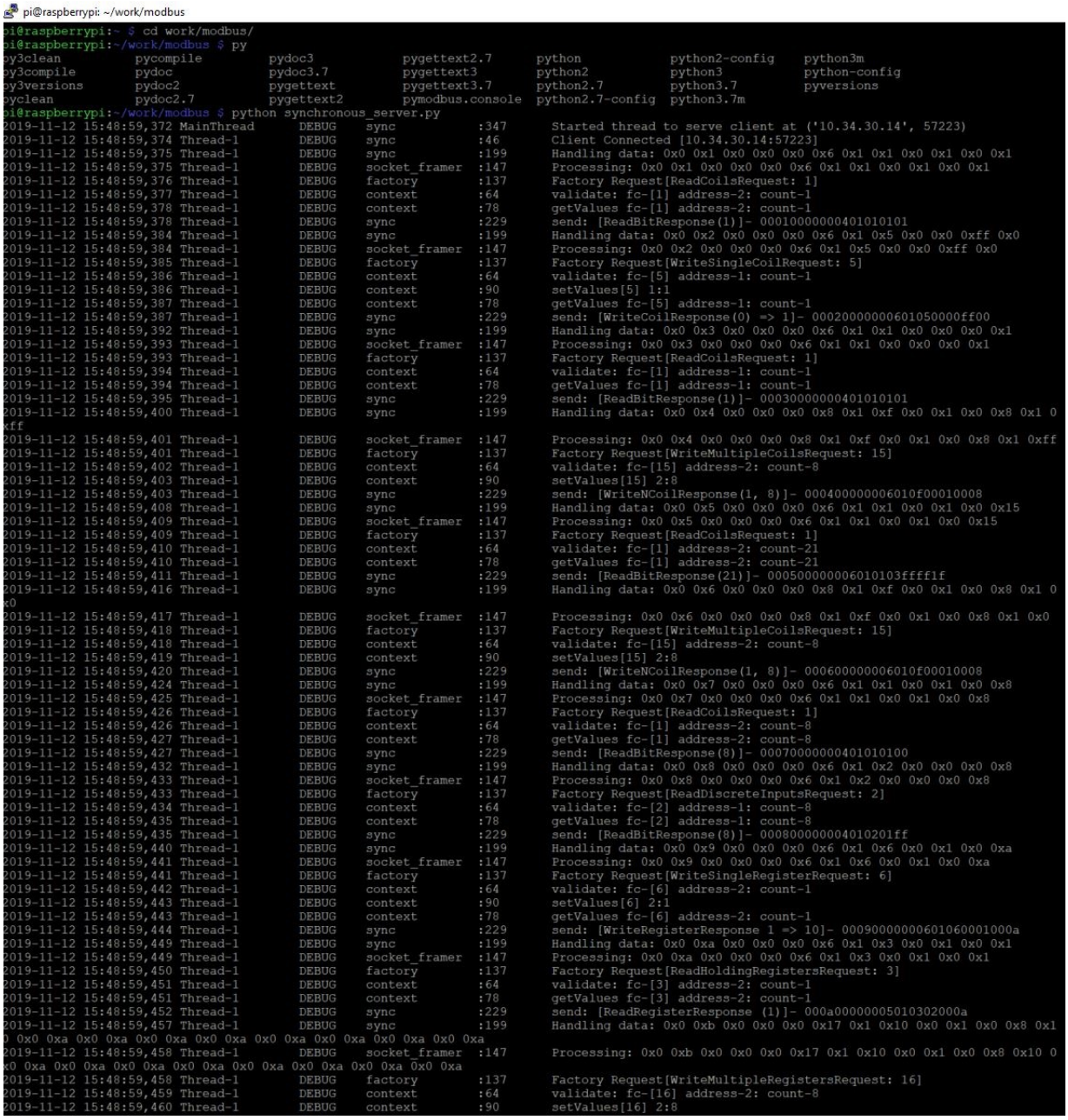
Obr. 2.1: Schéma zapojenia dvoch raspberry pi

Nasleduje príklad použitia synchronného klienta ModBus implementácia je z jazyka

Pythonu. Príklad výpisu pre ModBus ClientA.2. Realizácia bola možná s TCP aj UDP sú to vlastne protokoly používané na odosielanie bitov alebo inak povedané údajov tiež známych ako pakety ktoré sú posielané cez internet. V tomto prípade sme ModBus nakonfigurovali a realizovali cez protokol TCP. Na priložených obrázkoch z kódu vidno, že prenos bol jedno bitový a bolo prenášané 8 bitov označené to je červenou farbou. Kód sa dá podľa potreby samozrejme upravovať aj Klient taktiež server, ale na demonštráciu postačuje prenos týchto osem bitov.

2.1.2 Komunikácia ModBus v reálnej realizácii

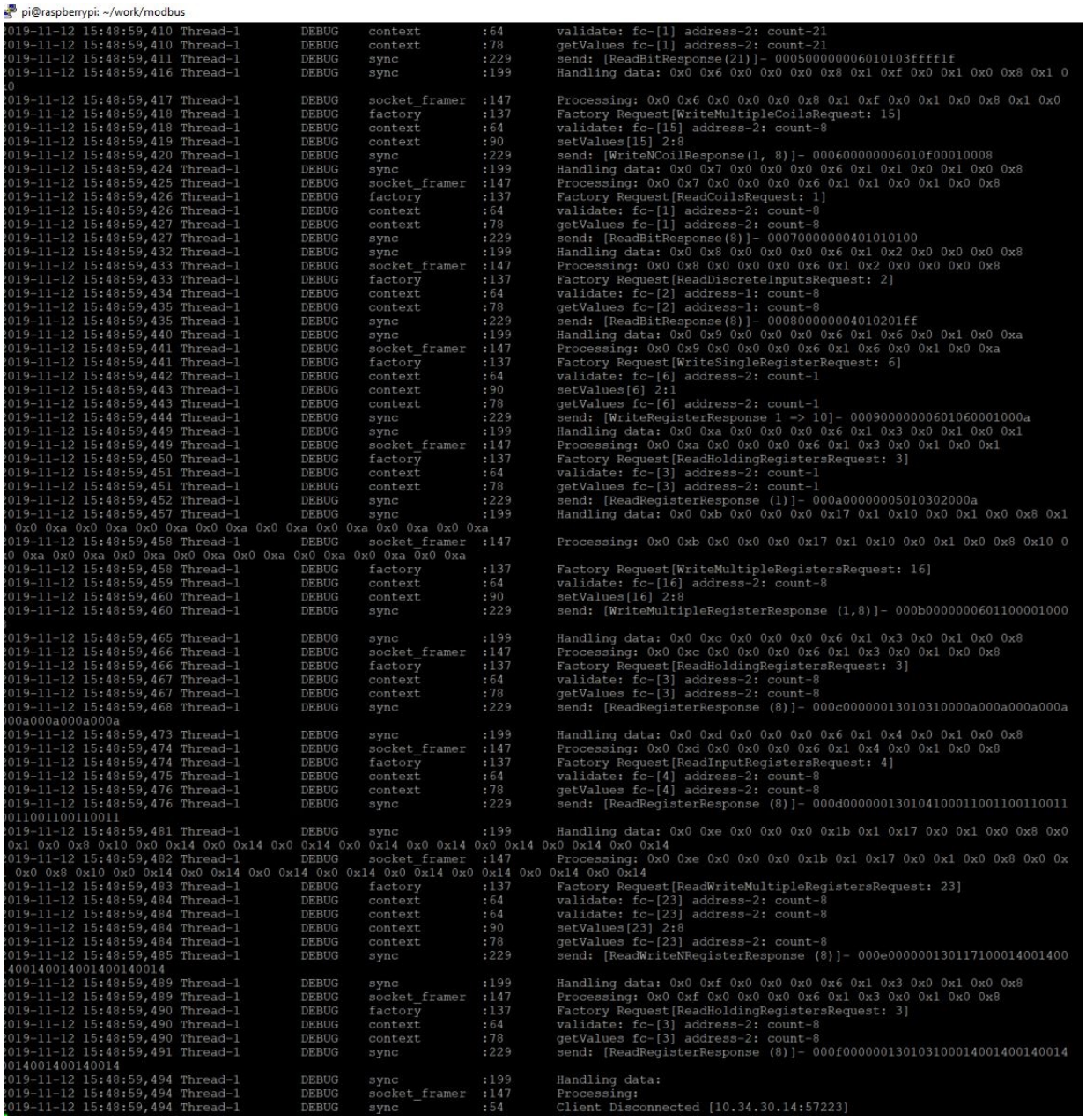
Realizácia v reálnych podmienkach bola uskutočnená otestovaním protokolu Modbus taktiež cez TCP kdeže tu musí byť prenos spoľahlivý. Uskutočnili sme to tak, že sme tiež spustili Server, následne sme s klientom sledovali čo sa bude diať. V našom príklade to bolo sledovanie tlaku a prepnutie medzi prvým, druhým režimom nastavenia stroja na obr. 2.6 je vidieť prvý režim červenou farbou a tlak 6,3 Bar zelenou farbou je to vlastne jeden bit určený pre čítanie. Následne sme sa pokúsili režim prepnúť a to nám zobrazuje obr. 2.7, na ktorom vidno červenou farbou, že režim sa zmenil a má to aj za následok pokles tlaku keďže v druhom režime je tlak využívaný. Ďalšiou nepostrehnutelnou súčasťou je, že na obrázkoch možno vidieť modrou farbou ako sa nám pakety postupným časom navyšujú je to následok toho, že komunikácia stále prebieha medzi serverom a klientom, ktorý nie je ukočený až



Obr. 2.2: Výpis z realizácie servera

po kým sa daný proces neukončí, alebo sa nestratí spojenie to by malo za následok vypnutie.

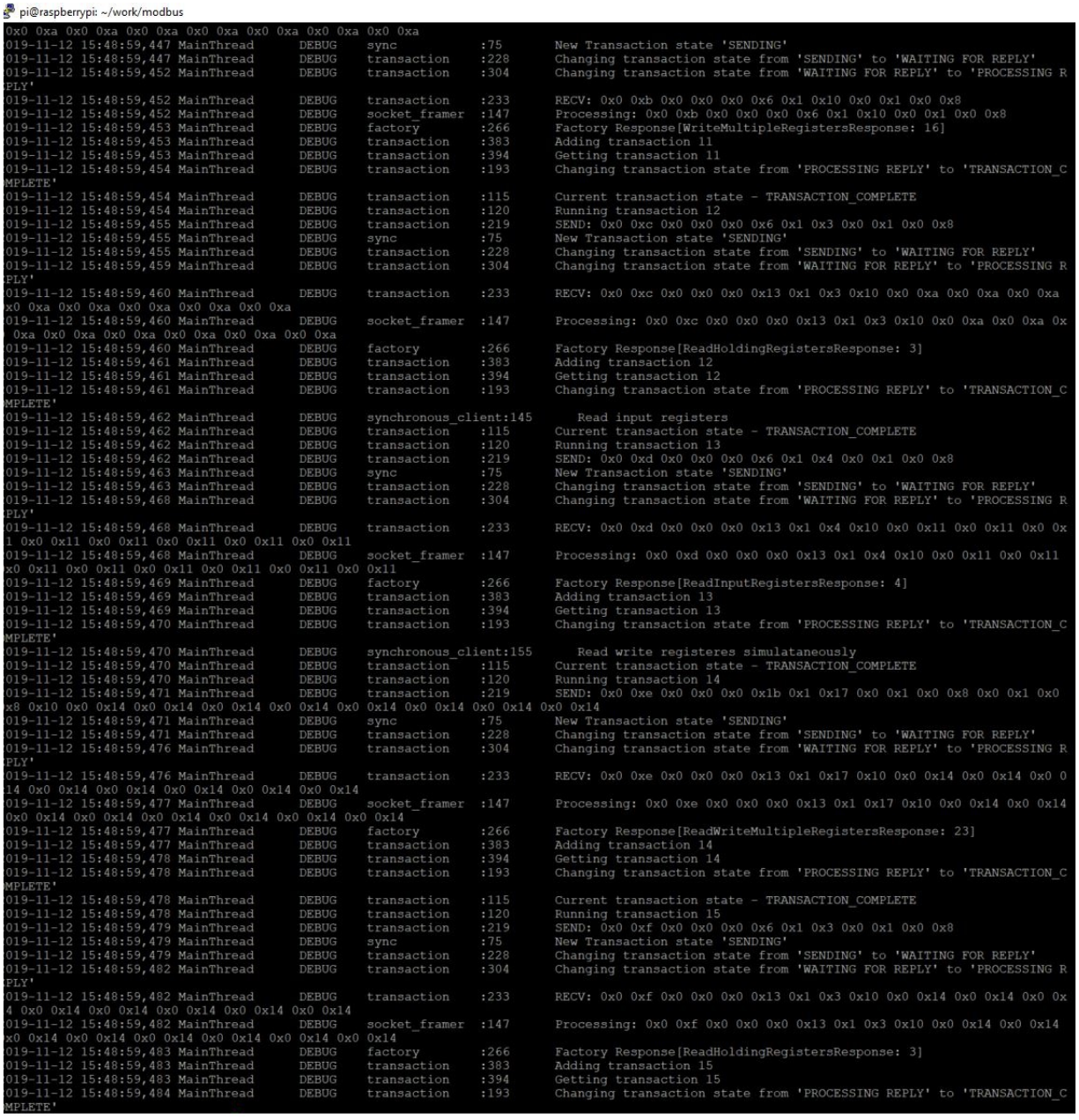
Ďalšiou možnosťou toho sledovania tejto komunikácie je Wireshark na obr. 2.8 vidno, že komunikácia prebieha stále a dokonca v priebehu času sa mení. Ďalšiou možnosťou je sledovanie vyťaženosť počítača v našom prípade máme počítač ktorý je na šťastie dostatočne výkonný, avšak keby to tak nebolo mohlo by sa stať, že nám to plne zahltí komunikáciu a ako vidno na obrázku obr. 2.8 by mohol nastať



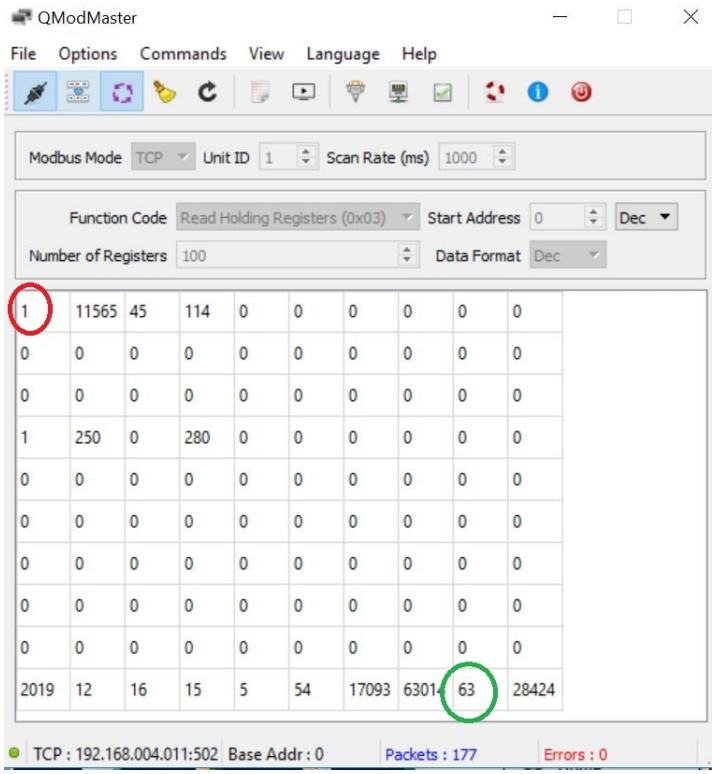
Obr. 2.3: Výpis z realizácie servera pokračovanie

prípad, že nemáme dostatočnú pamäť, alebo procesor obrázok obr. 2.9 na ľavej strane znázorňuje keď bol ModBus vypnutý a naopak na pravo keď bol zapnutý červenou farbou možno vidieť nárast vyťaženosti procesora a modrou nárast využiteľnosti.

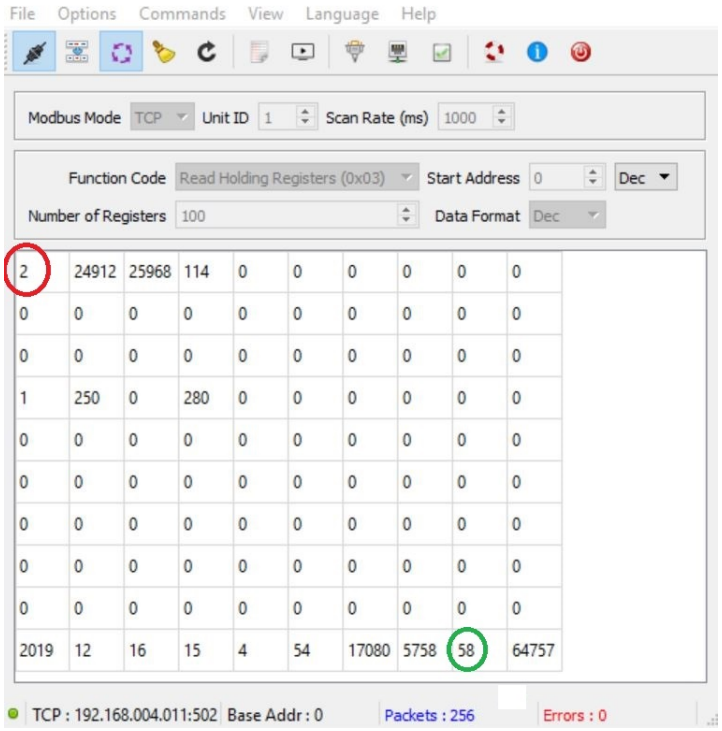
Obr. 2.4: Výpis z realizácie Klienta



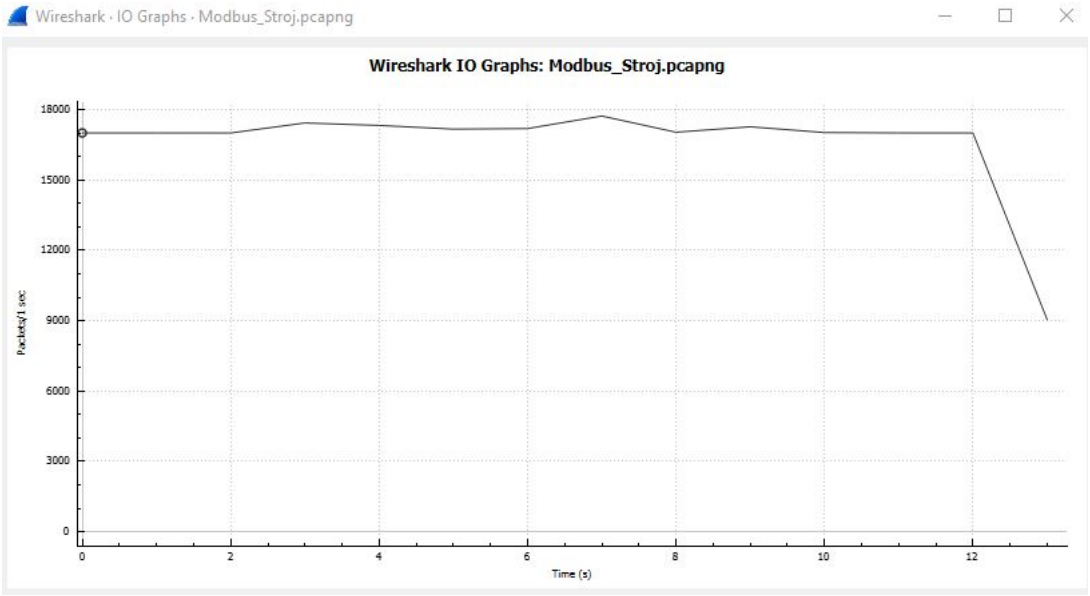
Obr. 2.5: Výpis z realizácie Klienta



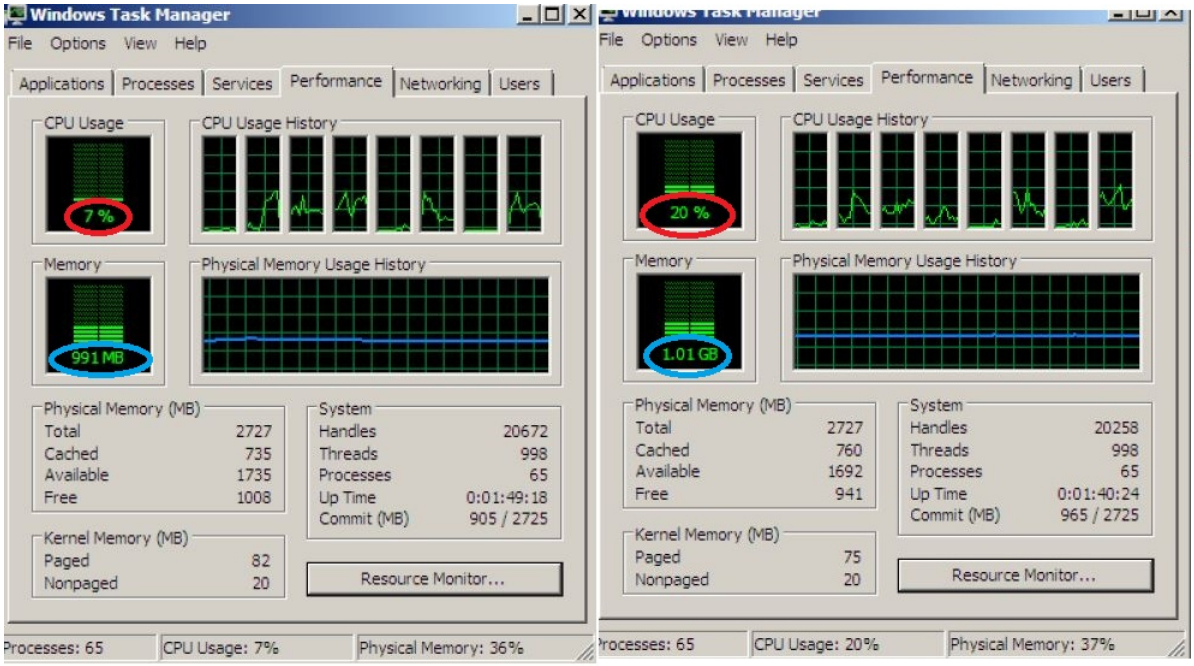
Obr. 2.6: ModBus Klient prvý režim



Obr. 2.7: ModBus Klient prepnutie režimu



Obr. 2.8: Sledovanie komunikácie vo Wiresharku



Obr. 2.9: Vyťaženosť počítača

2.2 Simulácia komunikácie Profinet

Pre dalšiu simuláciu komunikácie protokolu Profinet sme realizovali druhé pracovisko, ktoré bude simulovať sieťovú infraštruktúru priemyselných automatizačných protokolov. Ako druhý analyzovaný protokol sme zvolili priemyselný protokol Profinet.

Profinet nám umožňuje pripájať strojné zariadenia ako sú napríklad PLC a podobné zariadenia nazývané tiež ako vstupné/výstupné zariadenia Profinet, alebo podradené zariadenia k jednotke PLC to znamená k riadiacej jednotke vstupu a výstupu. Profinet ako hlavné zariadenie. Rozhranie Profinet IO ponúka cyklickú výmenu vstupných a výstupných údajových bitov na načítanie štatistických údajov o zariadení, na spustenie strojného zariadenia, alebo výber programu medzi jednotkou PLC a strojným zariadením. Rozhranie Profinet IO funguje použitím osobitného ethernetového pripojenia napríklad 100 Mbitov/s. Ethernetové rozhranie je vlastne medený párový skrúcaný kábel, konektor RJ45. Hlavnými rozdielmi medzi ModBusom a Profinetom sú rýchlosť a pripojiteľnosť.

2.2.1 Profinet realizácia s Raspberry Pi pomocou Scapy

Prvou realizáciu a skúšanie Profinetu sme realizovali pomocou Scapy ktorý je vlastne program napísaný v Pythone a užívateľovi umožňuje odosielať, čítať a falšovať sieťové pakety. Táto schopnosť umožňuje konštrukciu nástrojov, ktoré môžu snímať, skenovať alebo útočiť na siete. Inými slovami je Scapy výkonný interaktívny program na manipuláciu s paketmi. Je schopný falšovať alebo dekódovať pakety veľkého množstva protokolov čo nám dáva možnosť vyskúšať aj náš protokol Profinet pomocou Raspberry Pi. Ďalej môže zachytávať komunikáciu, odpovedať na žiadosti a odpovede a oveľa viac. Scapy dokáže ľahko zvládnuť väčšinu klasických úloh, ako sú skenovanie, sledovanie, testovanie, testovanie jednotiek, útoky alebo zisťovanie siete. Môže nahradiť hping, arpspoof, arp-sk, arping.

2.2.2 Profinet IO RTC

Na skúšku sme vybrali Profinet IO je priemyselný protokol, ktorý sa skladá z rôznych vrstiev, napríklad vrstvy RTC (Real-Time Cyclic), ktorá sa používa na výmenu údajov. Táto vrstva RTC je však stavová a závisí od konfigurácie odoslanej prostredníctvom inej vrstvy: koncového bodu DCE / RPC PROFINET. Táto konfigurácia definuje, kde každý vymenený údaj a ten sa musí nachádzať v RTC dátovej vyrovnávacej pamäti, ako aj dĺžku tej istej vyrovnávacej pamäte. Vytvorenie takého paketu je potom o niečo zložitejšie ako iné protokoly. Dátový paket RTC je to prvá vec, ktorú nám treba urobiť pri vytváraní vyrovnávacej pamäte údajov RTC, je nami vytvoriť

inštanciu každého paketu Scapy, ktorý predstavuje časť údajov. Každá z nich môže vyžadovať určitú konkrétnu konfiguráciu, napríklad jej dĺžku. Všetky pakety a ich konfigurácia sú:

- PNIORealTimeRawData to sú jednoduché nespracované údaje ako Raw
- Length definuje dĺžku údajov
- Profil PROFIsafe na zaistenie funkčnej bezpečnosti
- Tu Length definuje dĺžku celého paketu
- CRC definuje dĺžku CRC, buď 3 alebo 4
- PNIORealTimeIOxS buď bajt zákazníka IO alebo stav poskytovateľa

Na vytvorenie inštancie jedného z týchto paketov je potrebné zadať konfiguračný argument config. Je to dict(), ktorý obsahuje všetky požadované konfigurácie. Príklad zdrojového výpisu Raspberry Pi pre RTC data packet môžeme vidieť tu:B.1. Ďalej je možné teraz dátový paket inicializovať, možno tak zostaviť celý paket RTC. PNIORealTime obsahuje zozbierané údaje, ktoré sú zoznamom všetkých dátových paketov, ktoré sa majú pridať do medzipamäte, ale bez konfigurácie ich Scapy nebude môcť rozrezať. Príklad zdrojového výpisu Raspberry Pi pre PNIORealTime môžeme vidieť tu:B.9. Na to, aby ho Scapy mohla správne rozdeliť, je potrebné nakonfigurovať vrstvu tak, aby poznala umiestnenie všetkých údajov vo vyrovnávacej pamäti. Táto konfigurácia je uložená v slovníku conf.contribs["PNIO RTC"], ktorý je možné aktualizovať pomocou metódy pnio update config. Každá položka v slovníku používa ako kľúč n-ticu (Ether.src, Ether.dst), aby bolo možné oddeliť konfiguráciu každej komunikácie. Každá hodnota je potom zoznamom n-tice, ktorý popisuje dátový paket. Pozostáva z negatívneho indexu od konca vyrovnávacej pamäte údajov, polohy paketu, triedy paketu ako druhej položky a konfiguračného slovníka, ktorý triede poskytuje ako posledný. Ak budeme pokračovať v predchádzajúcom príklade, je potrebné nastaviť túto konfiguráciu. Tým pádom príklad zdrojového výpisu pre Raspberry Pi conf.contribs môžeme vidieť tu:B.4.

Ak pre daný offset nie sú nakonfigurované žiadne dátové pakety, štandardne sa použije PNIORealTimeIOxS. Táto metóda však nie je pre používateľa príliš výhodná na konfiguráciu vrstvy a ovplyvňuje iba rozrezanie paketov. V takýchto prípadoch môže mať prístup k viacerým paketom RTC, sledovaný alebo získaný zo súboru PCAP. PNIORealTime tak poskytuje niektoré metódy na analýzu zoznamu paketov PNIORealTime a na nájdenie všetkých údajov v ňom založených na jednoduchej heuristike. Všetci berú ako prvý argument iterovateľnosť, ktorá obsahuje zoznam paketov na analýzu.

- PNIORealTime.find data() analyzuje dátový buffer a oddeľuje skutočné údaje od IOxS. Vracia dielik, ktorý môže byť zadaný do pnio update config
- PNIORealTime.find profisafe() analyzuje dátovú vyrovnávaciu pamäť a vyhľadáva profily PROFIsafe medzi reálnymi údajmi. Vracia dielik, ktorý môže byť

zadaný do pnio update config.

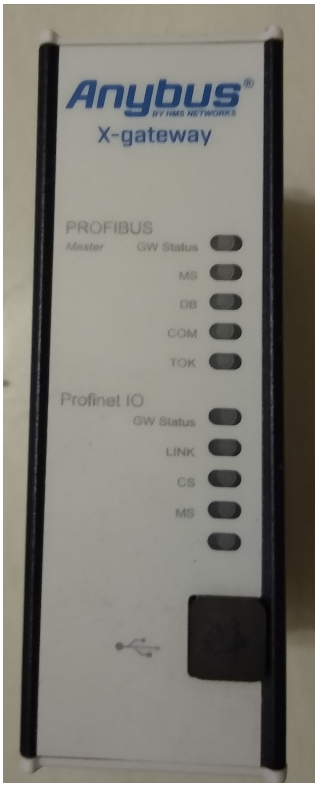
- PNIORealTime.analyse data() vykonáva predchádzajúce metódy a aktualizuje konfiguráciu. Toto je zvyčajne spôsob, ako zavolať túto metódu.
- PNIORealTime.draw entropy() nakreslí entropiu každého bajtu v dátovej vyrovnávacej pamäti. Môže sa použiť na jednoduchú vizualizáciu miest PROFIsafe, pretože entropia je základom rozhodovacieho algoritmu find profisafe.

Príklad tohto výpisu pre Raspberry Pi môžeme vidieť tu:B.6. Okrem toho je pri zobrazení paketu PNIORealTime vidieť len pole. Toto je vypočítané pole, ktoré nie je pridané v konečnom zostavení paketu. Je užitočný najmä pre disekciu a rekonštrukciu, ale môže sa taktiež použiť na zmenu správania paketu. V skutočnosti musí byť paket RTC vždy dostatočne dlhý pre ethernetový rámec a aby sa tak stalo, musí sa hneď po dátovej vyrovnávacej pamäti pridať výplň. Predvolené správanie je pridať výplň, ktorej veľkosť sa počíta počas procesu zostavovania. Príklad ako to vypadá pre Raspberry Pi si môžeme pozrieť tu:B.9. Je však možné nastaviť konfiguráciu (len) na zmenu tohto správania. Konfiguráciu (len) kontroluje tak dĺžku celého paketu PNIORealTime. Potom, aby sa skrátila dĺžka výplň, môže byť konfigurácia (len) nastavená na nižšiu hodnotu. Príklad pre Raspberry Pi si môžeme pozrieť tu:B.10.

2.2.3 Profinet realizácia s Raspberry Pi prostredníctvom Anybus X-gatway

Realizáciu na odskúšanie Profinetu sme realizovali pomocou PLC Siemens S7-1200 a brány Anybus X umožňuje nám bezproblémové prepojenie riadiacich systémov PLC a ich pripojených zariadení medzi sieťami Profibus a Profinet. Je to vlastne rýchle kopírovanie I/O údajov. Avšak primárnou funkciou X-brán je rýchly prenos cyklických I/O dát medzi oboma sieťami. To zbavuje PLC od práce s ďalšími výpočtami. Brána funguje ako slave v sieti Profibus a ako zariadenie v sieti Profinet. Prenos dát medzi oboma sieťami je úplne transparentný s maximálnou dátovou kapacitou 512 bajtov v každom smere. Jednoduchá konfigurácia a nie je nutné žiadne programovanie čo robí zariadenie pomerne jednoduchým. Spojenie medzi týmito dvoma sieťami sa rýchlo nastaví v softvéri Anybus Configuration Manager, ktorý je súčasťou X-brány. V predvolenom nastavení majú brány X preddefinovanú veľkosť I/O 20 bajtov I/O. My sme zvolili posielanie kde jednotlivé Signály sú Vstup sa rovná 1 bajt Výstup 1 bajt. Význam jednotlivých bitov 1-bajtový výstup z hlavného do podradeného zariadenia a 1-bajtový vstup z podradeného do hlavného zariadenia. Táto interpretácia je znázornéná na obr. 2.12 tu je interpretovaní 1 bit kde každý na niektorom mieste znamená niektorú zo zmien napríklad prvý bit je zmena funkčnosti stroja. Môžno to vidieť pre zmenu na obr. 2.13 kde bol zmenený bit kde nastalo

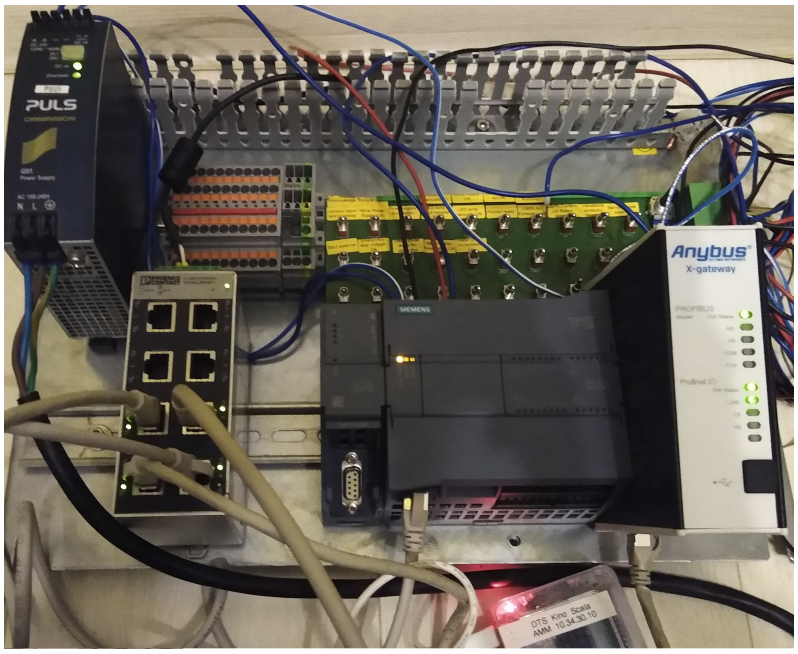
prepnutie čo by znamenalo spustenie stroja, ale zároveň aj pokles tlaku. Taktiež na obrázkoch vidno vstupy modrou farbou a výstupy červenou. Rozhranie Profinet ponúka takmer rovnaké vstupné a výstupné signály, ale vo forme logických bitov. Rozhranie Profinet tiež ponúka niektoré pulzné signály, čím sa odlišuje od štandardného základného reléového rozhrania! Pulzný režim sa používa na overenie signálov a kontrolu funkčnosti pripojeného strojného zariadenia. Logika pulzných signálov môže byť takáto, Aktívne to znamená zmenu hodnoty každú sekundu 0, 1, 0, 1, 0, 1, 0, 1, 0, 1... a tak ďalej. Neaktívne to znamená 0 konštantné, 0 alebo konštanta 1 po časovom limite 3 sekundy.



Obr. 2.10: Model ANYBUS

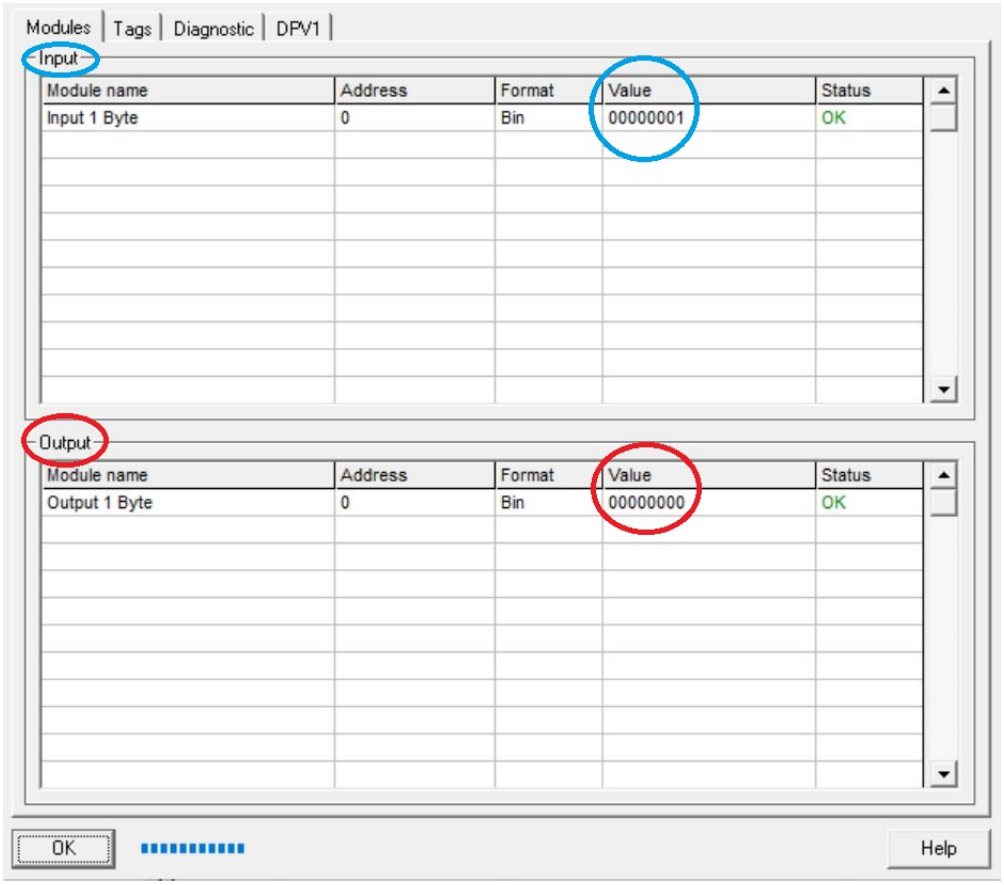
2.2.4 Profinet realizácia s Raspberry Pi s PLC Siemens S7-1200

Druhú realizáciu na odskúšanie Profinetu sme realizovali pomocou PLC Siemens S7-1200 a Raspberry Pi. PLC S7-1200 od Siemensu poskytuje flexibilitu a vysoký výkon na ovládanie širokej škály zariadení na podporu automatizačných potrieb čiže



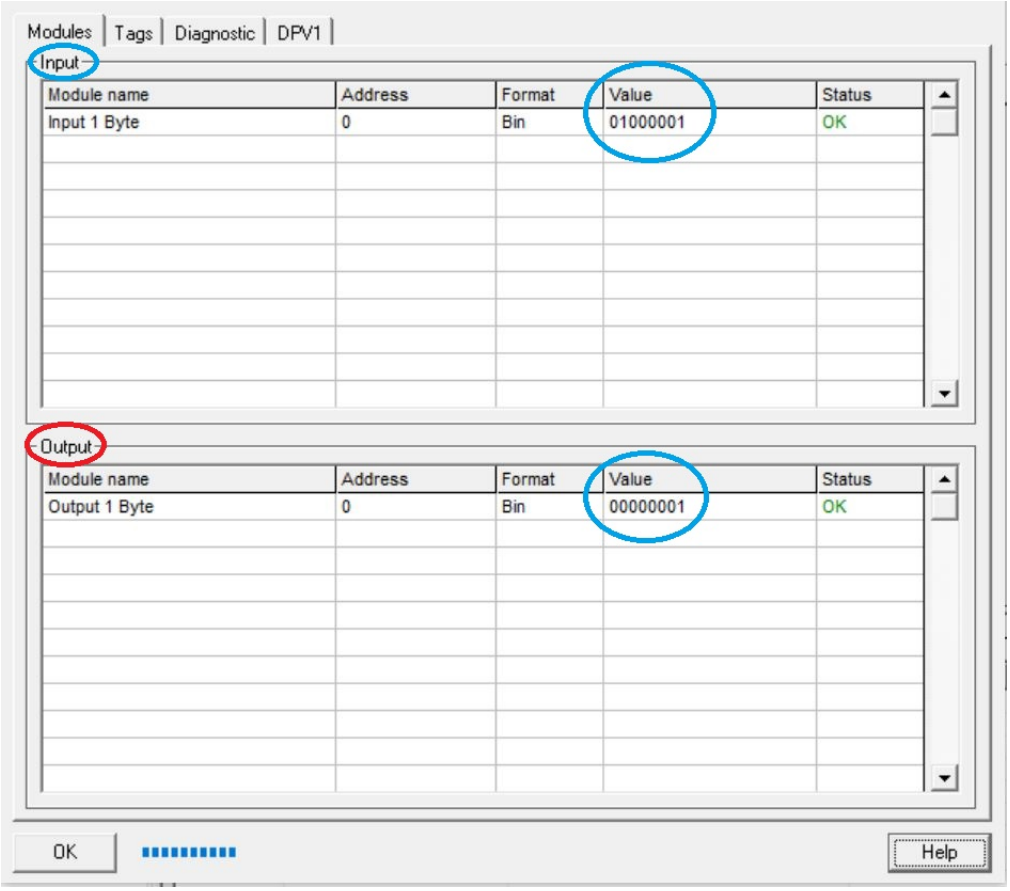
Obr. 2.11: Pripojenie Anybus Raspberry Pi

simuláciu protokolov respektíve ich používanie. Kompaktný dizajn a flexibilná konfigurácia je kombináciou výkonnej súpravy inštrukcií a S7-1200 sa stáva dokonalým riešením pre ovládanie a širokej škály aplikácií. CPU kombinuje mikroprocesor, integrovaný napájací, vstupný a výstupný obvod má vstavaný Profinet čo je pre nás idealne ďalej vysokorýchlostné I/O riadenie pohybu a zabudované analógové vstupy v kompaktnom prevedení čiže sa dá vytvoriť silný regulátor. Po stiahnutí programu sa CPU ľahko ovláda pretože obsahuje logiku potrebnú na monitorovanie a riadenie zariadení jednotlivých aplikácii. CPU monitoruje vstupy a zmeny výstupov podľa logiky užívateľského programu, ktorý môže zahŕňať logiku, počítanie, načasovanie, zložité matematické operácie a komunikáciu s inými inteligentnými zariadeniami. CPU poskytuje port Profinet pre komunikáciu cez sieť. K dispozícii sú ďalšie moduly na komunikáciu od Profibusu, GPRS, RS485 alebo Siete RS232 a iné. My sme pre našu realizáciu bude používať CPU 1214C DC/DC/RLY obr. 2.14. Napájanie je 24V čo nám poskytuje možnosť pripojiť aj ostatné zariadenia na jeden zdroj modul ktorý sme používali je možno vidieť na obr. 2.15. Ovládanie respektíve inštalácia je robená pomocou Totally Integrated Automation v skratke TIA-Portal. Pre riešenie jednotlivých automatizačných úloh je doteraz najviac osvedčený a je pre špecializované projekčné nástroje najleší. Jedná sa o Inžiniersky framework Totally Integrated Automation Portal (v skratke TIA-Portal) stiera hranicu medzi týmito nástrojmi. Tento inžiniersky koncept predstavuje nový základ pre vývojové nástroje na projektovanie, programovanie a uvádzanie do chodu veľa automatizačných prí-



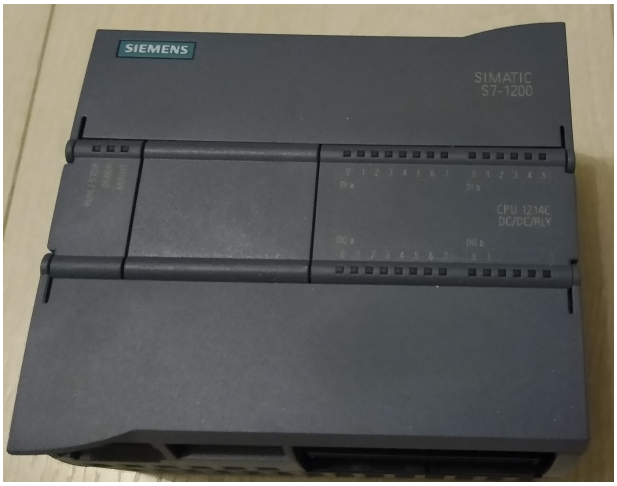
Obr. 2.12: Stav kľudu, vypnutý stav

strojov a pohonov. Prostredie možno vidieť na obr. 2.17. Tu je potreba nastavenie kde S7-1200 bude komunikovať s Raspberry Pi táto realizácia bola možná prostredníctvom ethernetu s RJ45 my sme zvolili CAT-6 pre lepšiu komunikáciu a pre lepšie tienenie. Pripojenie bolo treba nastaviť na S7-1200 aj na Raspberry Pi. Rasberry Pi bolo nastavené na posielanie jednotlivých bitov nastavenie a kód vidno naC.2 zmenou tohoto kódu je možno posielať jednotlivé biti. Konfiguráciu možno vidieť na obr. 2.14. Do siete bol naištalovaný aj prepínač plus riadiace PC, ktorým bolo možné ovládať S7-1200 a Raspberry Pi, to že komunikácia naozaj prebieha možno vidieť na obr. 2.16 na tomto obrázku tiež vidno, že zariadenie nie je až tak vyťažené to znamená, že hardvér CPU zvláda operácie a mohlo by ich byť oveľa viac. Schému tohoto pripojenia možno vidieť na obr. 2.19. Raspberry Pi nám v tomto prípade ako aj v predošlom poskytuje prenos a posielanie jednotlivých bitov táto komunikácia vypadala podobne ako bolo znázornené na obr. 2.12 a obr. 2.13 chovanie bolo rovnaké zmena kódu na Raspberry Pi to znamená posielanie bitu bolo viditeľné. Keďže je lepšie si overiť túto komunikáciu vyskúšali sme aj druhý softvér na sledovanie diania a na obr. 2.18 tu je vidno jednotlivé vstupy a výstupi ktoré sú označené

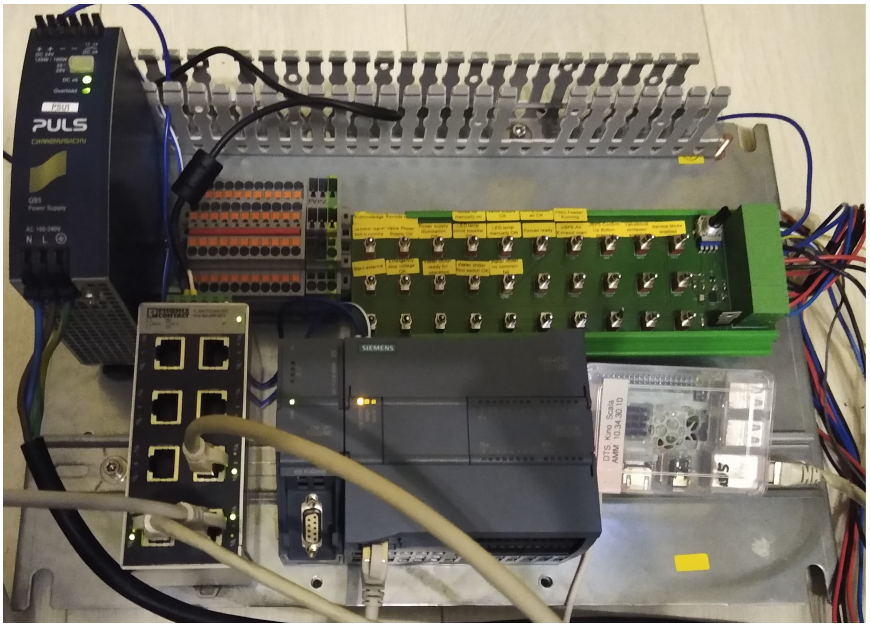


Obr. 2.13: Prepnutie stavu

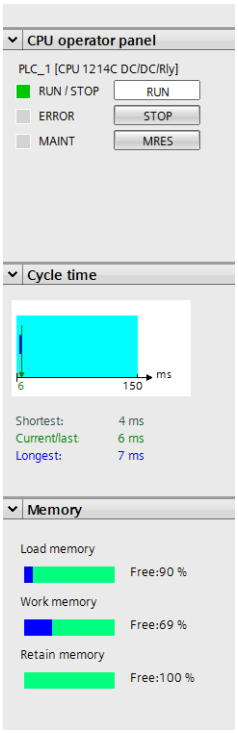
červenov farbou je označený vstup a naopak modrou výstup. V tomto režima ako vidno bol na určitom mieste bit tento bit konrétne znázorňoval chybu nastavená tlaku. Tento softvér sa nazýva Profinet master simulator a je to jednoduchý softvér na sledovanie výstupov a vstupov. Simulátory Profinet Master si môžu vymieňať údaje s mnohými zariadeniami. Okrem toho umožňujú simulátory Profinet Master umožnujú aj použitie súborov GSD a zadávanie špeciálnych konfigurácií na začatie výmeny údajov s podriadenými jednotkami Profinet. Je tiež možná identifikácia, premenovanie a priradenie adresy Profinetu. Aby sme vysvetlili tak obsah GSD pozostáva z konfiguračných informácií, parametrov, modulov, diagnostiky a alarmov a identifikácie výrobcu a zariadenia. Je to vlastne konfigurácia zariadenia, aby sa dalo poznať. Tretím zariadeným na odskúšanie Profinetu sme zvolili PLC allen bradley je to vlastne PLC využiváné prevažne v Amerike a na Americkom trhu. tu komikácia dopadla obdobne a posielanie jednotlivých bitov bolo tiež sprovoznené.



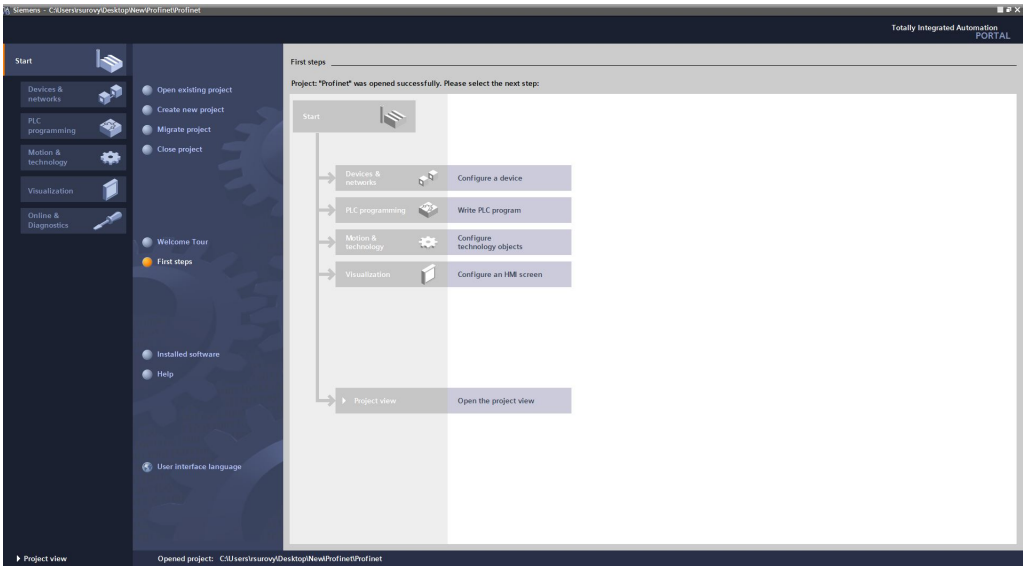
Obr. 2.14: Siemens S7-1200 model 1214C DC/DC/RLY



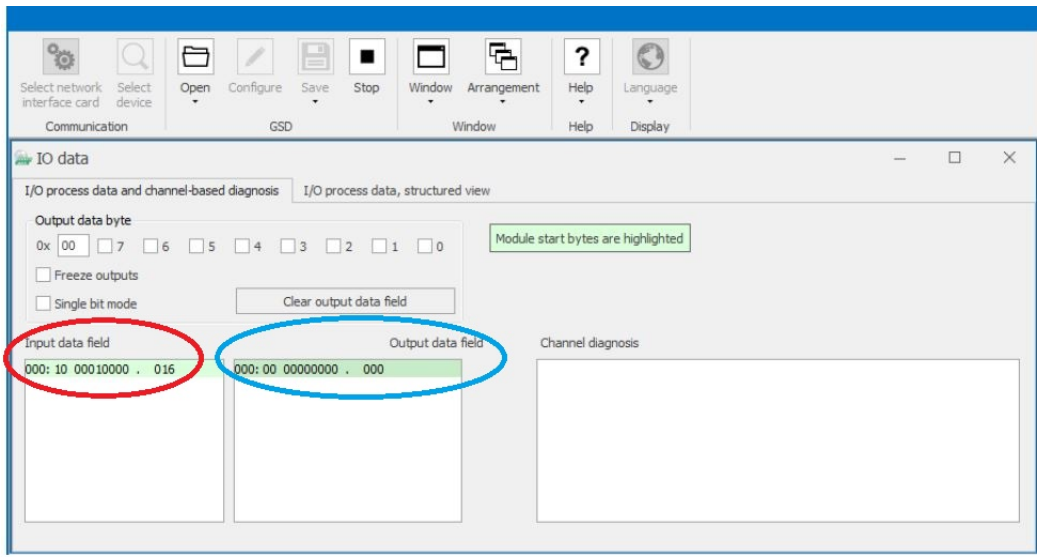
Obr. 2.15: Zapojenie Raspberry Pi a PLC S7-1200



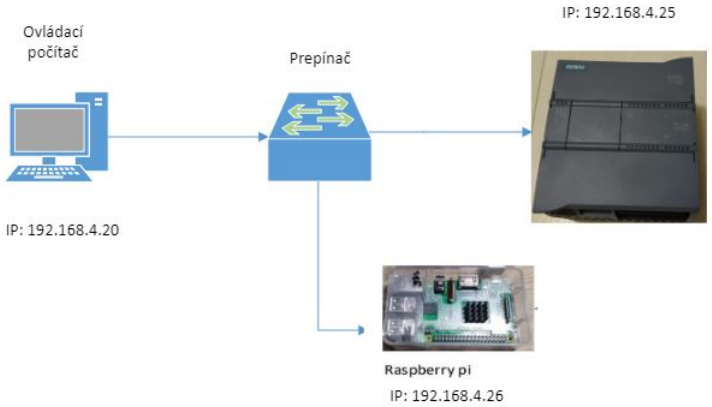
Obr. 2.16: Komunikácia PLC S7-1200 s Raspberry Pi



Obr. 2.17: Tia-Portal ukážka prostredia



Obr. 2.18: Profinet master simulator prezentácia výstupov a vstupov



Obr. 2.19: Schéma pripojenia PLC S7-1200 s Raspberry Pi

Záver

V tejto diplomovej práci sme si podrobne popísali automatizačné protokoly ModBus, Profibus/Profinet, EtherCAT a CANopen. Dokázali, sme že pomocou týchto protokolov sa dajú prenášať informácie a dáta ktoré môžu byť jednotlivými protokolmi chránené. Ďalej sme si ukázali ich najdôležitejšie vlastnosti použitie a využitie. Zrealizovali sme pracovisko, ktoré simulovalo sieťovú infraštruktúru dvoch vybraných priemyselných automatizačných protokolov. S pomedzi týchto všetkých protokolov sme si vybrali a otestovali ModBus a Profinet. Prvý ModBus je vlastne sériový komunikačný protokol. Jeho realizáciu sme uskutočnili prostredníctvom dvoch Raspberry Pi to nám nestačilo a zašli sme ďalej kde sme ho otestovali aj v reálnom prostredí kde bolo presne vidno načo sa dá protokol ModBus využiť reálnom prostredí. To znamená, že sa dá využiť na sledovanie napríklad tlaku, teploty, zapnutia, vypnutia stroja a mnoho ďalších. Zistili sme, že tento protokol bol publikovaný už v roku 1979 z toho nám vyplýva, že tento protokol je už dlho využívaný avšak používa. Sledovanie a jeho realizácia protokolu ModBus bola uskutočnená najskôr pomocou dvoch Raspberry pi. To znamená, že bolo potrebné využiť programovací jazyk a na túto konfiguráciu sme si vybrali pyhtona čo je interaktívny programovací jazyk. Keď bolo pracovisko zo sieťovou infraštruktúrou hotové odsledovali sme komunikáciu a následne sme ju aj zachytávali pomocou Wiresharku. Pri podobnej realizácii sme postupovali v reálnom prostredí kde sme prišli nato, že protokol je zastaralý a dal by sa nahradiť, pretože aj vyťažuje zariadenie ktoré je s ním spojené. To znamená, že je potreba o niečo výkonejší počítač a to či už to na pamäť, alebo procesor pretože slabšie zariadenie by nestíhalo.

Druhým analyzovaným protokolom s pomedzi vybraných bol Profinet. Na tomto protokole sme odsimulovali komunikáciu pomocou Scapy kde bolo vidno posielanie jednotlivých dát. Ďalšou skúškou Profinetu nám poskytovalo PLC S7-1200 kde bolo potreba nastaviť komunikáciu a veľa ďalších nastavení. Vyskúšali sme komunikáciu aj s AnyBusom a Allen bradley ktoré sa viac menej využíva prevažne na Americkom trhu. Zistili sme Profinet je moderný koncept distribuovaných automatizačných štandardov a je založený na ethernete čiže, integruje existujúce systémy priemyselnej zbernice a dokáže ich nielen sledovať čo sa vlastne deje, ale aj meniť stav podľa poslaných bitov, sledovanie výstupov a vstupov. ModBus je tiež skvelý protokol je využiteľný vo veľa smeroch, ale oproti Profinetu je výrazne zastaralí to znamená, že je náročnejší na hardvér čo bolo dokázané. Profinet nám naproti tomu výborne zvláda komunikáciu a to oveľa rýchlejšiu ako ModBus a samozrejme komunikáciu po ethernete čo je vo veľa prípadoch vynikajúca vlastnosť pretože sa s ním dá PLC ovládať na diaľku prostredníctvom aj vzdialenej komunikácie, alebo inak povedané internetu. To v praxi znamená, že ak prepojíme Profinet s VPN čo je vzdialená

komunikácia tak môžeme rôzne typy zariadení v našom prípade to bolo PLC to sa dá ovládať na diaľku čo nie je na škodu napríklad sa tým odstráni cena pri cestovaní a podobne. Vidno taktiež čo sa zo zariadením deje dá sa na týchto pozorovaniach vidieť množstvo štatistík.

Literatúra

[1] Modbus: The History of Modbus *Modbus History* [online]. 2016 [cit. 1996-11-11]. Dostupné z URL: <http://www.indusoft.com/blog/2016/11/11/the-history-of-modbus/>.

[2] Modbus: Modicon Modbus Protocol Reference Guide *Modbus Protocol* [online]. 1996 [cit. 1996-01-06]. Dostupné z URL: <http://www.modbus.org/docs/PI_MBUS_300.pdf>.

[3] Modbus: AboutModbus *Modbus* [online]. 2019 [cit. 2019-04-06]. Dostupné z URL: <https://www.simplymodbus.ca/FAQ.htm#Used>.

[4] Modbus: ModBus Communication Protocol *Modbus* [online]. 2019 [cit. 2019-05-15]. Dostupné z URL: <https://realpars.com/modbus-protocol/>.

[5] ModBus: ModBus tools *ModBus* [online]. 2019 [cit. 2019-04-04]. Dostupné z URL: <https://www.modbustools.com/modbus.html>.

[6] Profibus: Profibus Technology *Profibus* [online]. 2006 [cit. 2006-02-03]. Dostupné z URL: <https://us.profinet.com/technology/profibus//>.

[7] Profibus: Comprehensive Protocol Overview *Profibus* [online]. 2019 [cit. 2019-02-08]. Dostupné z URL: <https://www.rtautomation.com/technologies/profibus/>.

[8] Profibus: Profibus (Process Field Bus *Profibus* [online]. 2018 [cit. 2018-09-03]. Dostupné z URL: <https://www.kunbus.com/profibus-fieldbus.html>.

[9] Profibus: Profibus network communication protocol *Profibus* [online]. 2018 [cit. 2018-05-10]. Dostupné z URL: <https://visaya.solutions/en/article/everything-you-need-to-know-about-profibus>.

[10] Profibus: What is Profibus *Profibus* [online]. 2019 [cit. 2019-02-08]. Dostupné z URL: <http://www.smar.com/en/profibus>.

[11] Profinet: Profinet System Description *Profinet* [online]. 2014 [cit. 2014-01-10]. Dostupné z URL: <https://www.profibus.com/index.php?eID=dumpFile&t=f&f=51714&token=4ea5554cbb80a066e805a879116ead2a759c23c3>.

[12] Profinet IO: Profinet Unplugged – An introduction to PROFINET IO *Profinet IO* [online]. 2019 [cit. 2019-03-02]. Dostupné z URL: <https://www.rtautomation.com/technologies/profinet-io/>.

[13] Profinet: Profinet Communication Channels *Profinet* [online]. 2019 [cit. 2019-01-10]. Dostupné z URL: <https://profinetuniversity.com/profinet-basics/profinet-communication-channels/>.

[14] Profinet: Profinet Technology *Profinet* [online]. 2006 [cit. 2006-10-10]. Dostupné z URL: <https://us.profinet.com/technology/profinet/>.

[15] Profinet: What is Profinet *Profinet* [online]. 2019 [cit. 2019-01-10]. Dostupné z URL: <https://profinetuniversity.com/profinet-basics/definition-profinet/>.

[16] CANopen: History of CANopen *CANopen* [online]. 2004 [cit. 2004-07-10]. Dostupné z URL: <https://canopen.us/home/history-of-can>

[17] CANopen: CANopen Synopsis *CANopen* [online]. 2019 [cit. 2019-04-06]. Dostupné z URL: <https://www.rtautomation.com/technologies/canopen/>

[18] M. Farsi and K. Ratcliff, Än introduction to CANopen and CANopen communication issues,ÏEE Colloquium on CANopen Implementation (Digest No. 1997/384), London, UK, 1997, pp. 2/1-2/6. doi: 10.1049/ic:19971322 Dostupné z URL: <http://read.pudn.com/downloads367/doc/1590114/CANopen%20Implementation.pdf>

[19] A. Poschraann and P. Neumann, Ärchitecture and model of Profinet IO,"2004 IEEE Africon. 7th Africon Conference in Africa (IEEE Cat. No.04CH37590), Gaborone, 2004, pp. 1213-1218 Vol.2. doi: 10.1109/AFRICON.2004.1406884

[20] CANopen: The Basics of CANopen *CANopen* [online]. 2019 [cit. 2019-05-04]. Dostupné z URL: <https://www.ni.com/cs-cz/innovations/white-papers/13/the-basics-of-canopen.html>

[21] EtherCAT: What is EtherCAT *EtherCAT* [online]. 2019 [cit. 2019-07-08]. Dostupné z URL: <https://www.motioncontroltips.com/what-is-ethercat/>

[22] EtherCAT: WHAT IS ETHERCAT *EtherCAT* [online]. 2019 [cit. 2019-05-02]. Dostupné z URL: <https://realpars.com/ethercat/>

[23] Scapy:PROFINET IO RTC *Scapy* [online]. 2020 [cit. 2020-01-20]. Dostupné z URL: <https://scapy.readthedocs.io/en/latest/layers/pnio.html>

[24] Snap7: S7 Protocol (snap7) *Snap7* [online]. 2017 [cit. 2017-08-02]. Dostupné z URL: <http://www.simplyautomationized.blogspot.com/p/industrial-automation.html>

Zoznam symbolov, veličín a skratiek

CAN	Sieť riadiacej oblasti – Controller Area Network
CRC	Cyklická redundantní kontrola – Cyclic Redundancy Check
OSI	Opis návrhu štruktúry komunikačných a počítačových sieťových protokolov – Open Systems Interconnection Reference Model
PAC	programovateľné automatizačné ovládače
PLC	Programovatelný logický automat – Programmable Logic Controller
SCADA	Komplexný riadiaci a informačný systém – Supervisory Control and Data Aquisition system
TCP	Základný komunikačný protokol pre komunikáciu medzi počítačmi a aplikáciami po internete – Transmission Control Protocol
UDP	Uživateľský datagramový protokol – User Datagram Protocol
ZVEI	Centrálna asociácia pre elektrotechnický priemysel – Zentralverband Elektrotechnik und Elektroindustrie

Zoznam príloh

A Zdrojový kód pre ModBus

A.1 ModBus Server

Výpis A.1: Príklad zdrojového kódu pre ModBus Sever

```
// ModBus Server
#!/usr/bin/env python
"""
Pymodbus Synchronous Server Example

from pymodbus.server.sync import StartTcpServer
from pymodbus.server.sync import StartTlsServer
from pymodbus.server.sync import StartUdpServer
from pymodbus.server.sync import StartSerialServer

from pymodbus.device import ModbusDeviceIdentification
from pymodbus.datastore import ModbusSequentialDataBlock,
ModbusSparseDataBlock
from pymodbus.datastore import ModbusSlaveContext,
ModbusServerContext

from pymodbus.transaction import ModbusRtuFramer,
ModbusBinaryFramer
import logging
FORMAT = ('%(asctime)-15s %(threadName)-15s'
' %(levelname)-8s %(module)-15s:%(lineno)-8s
' %(message)s')
logging.basicConfig(format=FORMAT)
log = logging.getLogger()
log.setLevel(logging.DEBUG)

def run_server():
identity = ModbusDeviceIdentification()
identity.VendorName = 'Pymodbus'
identity.ProductCode = 'PM'
identity.VendorUrl='http://github.com/riptideio/pymodbus/'
identity.ProductName = 'Pymodbus Server'
identity.ModelName = 'Pymodbus Server'
identity.MajorMinorRevision = '2.3.0'

if __name__ == "__main__":
run_server()
```

A.2 ModBus `Client`

Výpis A.2: Príklad zdrojového kódu pre ModBus Client

```
// ModBUS Client
  #!/usr/bin/env python
"""
Pymodbus␣Synchronous␣Client␣Examples
"""
from pymodbus.client.sync
import ModbusTcpClient as ModbusClient
import logging
FORMAT = ('%(asctime)-15s␣%(threadName)-15s␣'
'%(levelname)-8s␣%(module)-15s:
'%(lineno)-8s %(message)s')
logging.basicConfig(format=FORMAT)
log␣=␣logging.getLogger()
log.setLevel(logging.DEBUG)
UNIT␣=␣0x1
def␣run_sync_client():
client␣=␣ModbusClient('localhost',␣port=5020)
client.connect()
log.debug("Reading␣Coils")
rr␣=␣client.read_coils(1,␣1,␣unit=UNIT)
log.debug(rr)

log.debug("Write␣to␣a␣Coil␣and␣read␣back")
rq␣=␣client.write_coil(0,␣True,␣unit=UNIT)
rr␣=␣client.read_coils(0,␣1,␣unit=UNIT)
assert(not␣rq.isError())
assert(rr.bits[0]␣==␣True)

log.debug("Write␣to␣multiple␣coils␣and␣read␣back-␣test␣1")
rq␣=␣client.write_coils(1,␣[True]*8,␣unit=UNIT)
assert(not␣rq.isError())
rr␣=␣client.read_coils(1,␣21,␣unit=UNIT)
assert(not␣rr.isError())

resp␣=␣[True]*21
resp.extend([False]*3)
assert(rr.bits␣==␣resp)
log.debug("Write␣to␣multiple␣coils␣and␣read␣back␣-␣test␣2")
```

```
␣␣                                                                          39
rq␣=␣client.write_coils(1,␣[False]*8,␣unit=UNIT)                            40
rr␣=␣client.read_coils(1,␣8,␣unit=UNIT)                                     41
assert(not␣rq.isError())                                                    42
assert(rr.bits␣==␣[False]*8)                                                43
␣␣                                                                          44
log.debug("Read␣discrete␣inputs")                                           45
rr␣=␣client.read_discrete_inputs(0,␣8,␣unit=UNIT)                           46
assert(not␣rq.isError())                                                    47
                                                                            48
log.debug("Write␣to␣a␣holding␣register␣and␣read␣back")                       49
rq␣=␣client.write_register(1,␣10,␣unit=UNIT)                                50
rr␣=␣client.read_holding_registers(1,␣1,␣unit=UNIT)                         51
assert(not␣rq.isError())                                                    52
assert(rr.registers[0]␣==␣10)                                               53
                                                                            54
log.debug("Write␣to␣multiple␣holding␣registers␣and␣read␣back")              55
rq␣=␣client.write_registers(1,␣[10]*8,␣unit=UNIT)                           56
rr␣=␣client.read_holding_registers(1,␣8,␣unit=UNIT)                         57
assert(not␣rq.isError())                                                    58
assert(rr.registers␣==␣[10]*8)                                              59
                                                                            60
log.debug("Read␣input␣registers")                                           61
rr␣=␣client.read_input_registers(1,␣8,␣unit=UNIT)                           62
assert(not␣rq.isError())                                                    63
arguments␣=␣{                                                               64
'read_address':␣␣␣␣1,                                                       65
'read_count':␣␣␣␣␣␣␣8,                                                      66
'write_address':␣␣␣1,                                                       67
'write_registers':␣[20]*8,                                                  68
␣}                                                                          69
log.debug("Read␣write␣registeres␣simulataneously")                          70
rq␣=␣client.readwrite_registers(unit=UNIT,␣**arguments)                     71
rr␣=␣client.read_holding_registers(1,␣8,␣unit=UNIT)                         72
assert(not␣rq.isError())                                                    73
assert(rq.registers␣==␣[20]*8)                                              74
assert(rr.registers␣==␣[20]*8)                                              75
client.close()                                                              76
if␣__name__␣==␣"__main__":                                                  77
run_sync_client()                                                           78
```

!

B Zdrojový kód pre Profinet

B.1 Profinet `RTC data packet`

Výpis B.1: Príklad tohto výpisu a zdrojového kódu Raspberry Pi pre RTC data packet môžeme vidieť tu:

```
// RTC data packet
>>>load_contrib('pnio_rtc')
>>>raw(PNIORealTimeRawData(load='AAA', config={'length': 4}))
b'AAA\x00'
>>>raw(Profisafe(load='AAA', Control_Status=0x20,
CRC=0x424242, config={'length': 8, 'CRC': 3}))
'AAA\x00␣BBB'
>>>hexdump(PNIORealTimeIOxS())
0000   80
```

B.2 RTC packet `PNIORealTime`

Výpis B.2: Príklad tohto výpisu a zdrojového kódu Raspberry Pi pre PNIORealTime môžeme vidieť tu:

```
// PNIORealTime
>>> load_contrib("pnio_rtc")
>>> p=PNIORealTime(cycleCounter=1024, data=[
... PNIORealTimeIOxS(),
... PNIORealTimeRawData(load='AAA',
config={'length':4}) / PNIORealTimeIOxS(),
... Profisafe(load='AAA', Control_Status=0x20, CRC=0x424242,
config={'length': 8, 'CRC': 3}) / PNIORealTimeIOxS(),
... ])
>>> p.show()
###[ PROFINET Real-Time ]###
  len= None
  dataLen= None
  \data\
   |###[ PNIO RTC IOxS ]###
   |  dataState= good
   |  instance= subslot
   |  reserved= 0x0
   |  extension= 0
   |###[ PNIO RTC Raw data ]###
   |  load= 'AAA'
   |###[ PNIO RTC IOxS ]###
   |     dataState= good
   |     instance= subslot
   |     reserved= 0x0
   |     extension= 0
   |###[ PROFISafe ]###
   |  load= 'AAA'
   |  Control_Status= 0x20
   |  CRC= 0x424242
   |###[ PNIO RTC IOxS ]###
   |     dataState= good
   |     instance= subslot
   |     reserved= 0x0
   |     extension= 0
  padding= ''
  cycleCounter= 1024
  dataStatus= primary+validData+run+no_problem
  transferStatus= 0
```

B.3 RTC packet `PNIORealTime pokračovanie`

Výpis B.3: Príklad tohto výpisu a zdrojového kódu Raspberry Pi pre PNIORealTime pokračovanie môžeme vidieť tu:

```
// PNIORealTime pokračovanie
>>> p.show2()
###[ PROFINET Real-Time ]###
  len= 44
  dataLen= 15
  \data\
   |###[ PNIO RTC Raw data ]###
   |  load= '\x80AAA\x00\x80AAA\x00␣BBB\x80'
  padding= ''
  cycleCounter= 1024
  dataStatus= primary+validData+run+no_problem
  transferStatus= 0
```

B.4 RTC packet `conf.contribs`

Výpis B.4: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi conf.contribs môžeme vidieť tu:

```
// conf.contribs
>>> load_contrib("pnio")
>>> e=Ether(src='00:01:02:03:04:05',
dst='06:07:08:09:0a:0b') / ProfinetIO() / p
>>> e.show2()
###[ Ethernet ]###
  dst= 06:07:08:09:0a:0b
  src= 00:01:02:03:04:05
  type= 0x8892
###[ ProfinetIO ]###
     frameID= RT_CLASS_1
###[ PROFINET Real-Time ]###
  len= 44
  dataLen= 15
  \data\
   |###[ PNIO RTC Raw data ]###
   |  load= '\x80AAA\x00\x80AAA\x00␣BBB\x80'
  padding= ''
  cycleCounter= 1024
  dataStatus= primary+validData+run+no_problem
  transferStatus= 0
>>> pnio_update_config({('00:01:02:03:04:05',
'06:07:08:09:0a:0b'): [
... (-9, Profisafe, {'length': 8, 'CRC': 3}),
... (-9 - 5, PNIORealTimeRawData, {'length':4}),
... ]})
>>> e.show2()
###[ Ethernet ]###
  dst= 06:07:08:09:0a:0b
  src= 00:01:02:03:04:05
  type= 0x8892
###[ ProfinetIO ]###
     frameID= RT_CLASS_1
```

B.5 RTC packet conf.contribs pokračovanie

Výpis B.5: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi conf.contribs pokračovanie môžeme vidieť tu:

```
// conf.contribs pokračovanie
###[ PROFINET Real-Time ]###
        len= 44
        dataLen= 15
        \data\
         |###[ PNIO RTC IOxS ]###
         |  dataState= good
         |  instance= subslot
         |  reserved= 0x0L
         |  extension= 0L
         |###[ PNIO RTC Raw data ]###
         |  load= 'AAA'
         |###[ PNIO RTC IOxS ]###
         |     dataState= good
         |     instance= subslot
         |     reserved= 0x0L
         |     extension= 0L
         |###[ PROFISafe ]###
         |  load= 'AAA'
         |  Control_Status= 0x20
         |  CRC= 0x424242L
         |###[ PNIO RTC IOxS ]###
         |     dataState= good
         |     instance= subslot
         |     reserved= 0x0L
         |     extension= 0L
        padding= ''
        cycleCounter= 1024
        dataStatus= primary+validData+run+no_problem
        transferStatus= 0
```

B.6 RTC packet PNIORealTimeIOxS

Výpis B.6: Príklad tohto výpisu pre Raspberry Pi môžeme vidieť tu:

```
// PNIORealTimeIOxS
>>> load_contrib('pnio_rtc')
>>> t=rdpcap('/path/to/trace.pcap', 1024)
>>> PNIORealTime.analyse_data(t)
{('00:01:02:03:04:05', '06:07:08:09:0a:0b'):
[(-19, <class 'scapy.contrib.pnio_rtc.PNIORealTimeRawData'>,
{'length': 1}),
(-15, <class 'scapy.contrib.pnio_rtc.Profisafe'>,
{'CRC': 3, 'length': 6}),
(-7, <class 'scapy.contrib.pnio_rtc.Profisafe'>,
{'CRC': 3, 'length': 5})]}
>>> t[100].show()
###[ Ethernet ]###
  dst= 06:07:08:09:0a:0b
  src= 00:01:02:03:04:05
  type= n_802_1Q
###[ 802.1Q ]###
     prio= 6L
     id= 0L
     vlan= 0L
     type= 0x8892
###[ ProfinetIO ]###
        frameID= RT_CLASS_1
###[ PROFINET Real-Time ]###
           len= 44
           dataLen= 22
           \data\
            |###[ PNIO RTC Raw data ]###
            |  load= '\x80\x80\x80\x80\x80\x80
\x00\x80\x80\x80\x12:\x0e\x12\x80
\x80\x00\x12\x8b\x97\xe3\x80'
           padding= ''
           cycleCounter= 6208
           dataStatus= primary+validData+run+no_problem
           transferStatus= 0
>>> t[100].show2()
###[ Ethernet ]###
  dst= 06:07:08:09:0a:0b
  src= 00:01:02:03:04:05
  type= n_802_1Q
```

B.7 RTC packet PNIORealTimeIOxS pokračovanie

Výpis B.7: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi pre PNIORealTimeIOxS pokračovanie môžeme vidieť tu:

```
// PNIORealTimeIOxS pokračovanie
###[ 802.1Q ]###
     prio= 6L
     id= 0L
     vlan= 0L
     type= 0x8892
###[ ProfinetIO ]###
        frameID= RT_CLASS_1
###[ PROFINET Real-Time ]###
           len= 44
           dataLen= 22
           \data\
            |###[ PNIO RTC IOxS ]###
            |  dataState= good
            |  instance= subslot
            |  reserved= 0x0L
            |  extension= 0L
            [...]
            |###[ PNIO RTC IOxS ]###
            |  dataState= good
            |  instance= subslot
            |  reserved= 0x0L
            |  extension= 0L
            |###[ PNIO RTC Raw data ]###
            |  load= ''
            |###[ PNIO RTC IOxS ]###
            |     dataState= good
            |     instance= subslot
            |     reserved= 0x0L
            |     extension= 0L
            [...]
            |###[ PNIO RTC IOxS ]###
            |  dataState= good
            |  instance= subslot
            |  reserved= 0x0L
            |  extension= 0L
```

B.8 RTC packet `PNIORealTimeIOxS druhé pokračovanie`

Výpis B.8: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi pre PNIORealTimeIOxS druhé pokračovanie môžeme vidieť tu:

```
// PNIORealTimeIOxS druhé pokračovanie

            |###[ PROFISafe ]###
            |  load= ''
            |  Control_Status= 0x12
            |  CRC= 0x3a0e12L
            |###[ PNIO RTC IOxS ]###
            |     dataState= good
            |     instance= subslot
            |     reserved= 0x0L
            |     extension= 0L
            |###[ PNIO RTC IOxS ]###
            |  dataState= good
            |  instance= subslot
            |  reserved= 0x0L
            |  extension= 0L
            |###[ PROFISafe ]###
            |  load= ''
            |  Control_Status= 0x12
            |  CRC= 0x8b97e3L
            |###[ PNIO RTC IOxS ]###
            |     dataState= good
            |     instance= subslot
            |     reserved= 0x0L
            |     extension= 0L
          padding= ''
          cycleCounter= 6208
          dataStatus= primary+validData+run+no_problem
          transferStatus= 0
```

B.9 RTC packet `PNIORealTime`

Výpis B.9: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi si môžeme pozrieť tu:

```
// PNIORealTime
>>> raw(PNIORealTime(cycleCounter=0x4242,
data=[PNIORealTimeIOxS()]))
'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00BB5\x00'
```

B.10 RTC packet `len`

Výpis B.10: Príklad tohto výpisu a zdrojového kódu Raspberry Pi si môžeme pozrieť tu:

```
// len
>>> raw(PNIORealTime(cycleCounter=0x4242,
data=[PNIORealTimeIOxS()], len=50))
'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00BB5\x00'
>>> raw(PNIORealTime(cycleCounter=0x4242,
data=[PNIORealTimeIOxS()]))
'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00BB5\x00'
>>> raw(PNIORealTime(cycleCounter=0x4242,
data=[PNIORealTimeIOxS()], len=30))
'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00BB5\x00'
```

C Zdrojový kód pre snap7-python

C.1 Snap7-python `Snap7`

Výpis C.1: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi inštálacia sna7 môžeme vidieť tu:

```
// Snap7 Installation
$ sudo add-apt-repository ppa:gijzelaar/snap7
$ sudo apt-get update
$ sudo apt-get install libsnap71 libsnap7-dev

$ p7zip -d snap7-full-1.0.0.7z
# requires the p7 program
$ cd build/<platform>
# where platform is unix or windows
$ make -f <arch>.mk
# where arch is your architecture, for example x86_64_linux

$ pip install python-snap7

$ python ./setup.py install
```

Výpis C.2: Príklad tohto výpisu a zdrojového kódu pre Raspberry Pi posielanie bitov:

```
// Snap7 bit
import snap7.client as c
 from snap7.util import *
 from time import sleep

def WriteOutput(dev,bytebit,cmd):
     byte,bit = bytebit.split('.')
     byte,bit = int(byte),int(bit)
     data = dev.read_area(0x82,0,byte,1)
     set_bool(data,byte,bit,cmd)
     dev.write_area(0x82,byte,data)

 def main():
     myplc = snap7.client.Client()
     myplc.connect('192.168.4.26',0,1)
     for x in range(10):
         WriteOutput(myplc,'0.0',x%2==0)
# turns true every other iteration
         sleep(1)

 if __name__ == "__main__":
     main()
```